gRNA

February 12, 2020

0.1 Guide RNA design using Geckov2 and Broad datasets for library construction

- Join datasets (Broad and Geck).
- Count the sequences per DNA symbol
- Create a dataset with max six sequences.

0.2 Importing packages

[590]:
```
#============================================================
# Packages
#============================================================

import pandas as pd
import numpy as np
import os # Accesing to directory
import re # Regular Expressions
from six.moves import reduce # Merge dataframes


## Setting the seed value for reproducibility

seed_value= 123# Set a seed value

# Set `python` built-in pseudo-random generator at a fixed value
import random
random.seed(seed_value)

# Set `numpy` pseudo-random generator at a fixed value
import numpy as np
np.random.seed(seed_value)

seed =  np.random.RandomState(123)
# do not call numpy.random.rand() but seed.rand()

# 3. Set environment
os.urandom(seed_value)
```

[590]:
```
b'[tgpu\xb5+6\x13\xb6<\xfc\xc7`\x11\xb2\xfd9\xc1o\xd5WC=\x13>\xd7Q\xc14\x0e\r\xd
5\x9b\xd2hV\xfd2\x0b\xecgiv:\x9a\xbc\xf0\t\xce.\xad\x93~\xc6\xa6\x13Z\xde\xfe\xe
3\x1fY\x93?\xa2SZ\x914\x16\xd67\x9eL\xb7$.\xe7\x10[\xfd\x1b\x8cC\xd8\xb5\xf5\x14
W\xcb\xe5\xc7\xc4\xa8U\x95$XPh\xaec\xe7\xfe\xba\xe7&\x00\x82\x1d\xf4L\xd1x\xccE\
xafn\x8d\xe4n\x93'
```

0.3 Defining Functions

- Defining function to be used in this script
- Cleaning gRNA sequences

[688]:
```
#============================================================
# Reading Files
#============================================================

def read_files(file_path):
    '''
    Function that reads two files as dataframes, and rename the two columns.
    Output = a dataframes
    '''
    df_file = pd.read_csv(file_path) #delimiter="\t"
    columns_names = ['gene_symbol', 'sequence']
    df_file.columns = columns_names

    return df_file


#============================================================
# Cleaning Files
#============================================================

def removing_characters(df):
    ''' Here are the rules of designing the gRNA on our end:
    1. gRNA can't contain CACCTGC or GCAGGTG or CGTCTC or GAGACG or CTCGAG
    2. The first 6 bases can't be CAGGTG; The first 5 bases can't be AGACG;
       The first 4 bases can't be TCTC
    3. The last 6 bases can't be GCAGGT; The last 5 bases can't be GAGAC or␣
 ↪CTCGA
    4. The length of all gRNAs is desired to be the same.

    input/output file is a dataframe
    '''
    for column in df.columns:
        if column in ['sequence']:
            print (column)
```

```
            # gRNA can't contain CACCTGC or GCAGGTG or CGTCTC or GAGACG or␣
↪CTCGAG
            #df[column] = df[column].apply(Clean_seq)
            pattern = ['CACCTGC','GCAGGTG', 'CGTCTC', ' GAGACG', 'CTCGAG']

            df.loc[df[column].str.contains('CACCTGC', case=False, na=False)] =␣
↪np.nan
            df.loc[df[column].str.contains('GCAGGTG', case=False, na=False)] =␣
↪np.nan
            df.loc[df[column].str.contains('CGTCTC', case=False, na=False)] =␣
↪np.nan
            df.loc[df[column].str.contains('GAGACG', case=False, na=False)] =␣
↪np.nan
            df.loc[df[column].str.contains('CTCGAG', case=False, na=False)] =␣
↪np.nan


            #The first 6 bases can't be CAGGTG;
            df.loc[df[column].str.startswith('CAGGTG', na=False)] = np.nan

            #The first 5 bases can't be AGACG;
            df.loc[df[column].str.startswith('AGACG', na=False)] = np.nan

            #The first 4 bases can't be TCTC
            df.loc[df[column].str.startswith('TCTC', na=False)] = np.nan

            # The last 6 bases can't be GCAGGT;
            df.loc[df[column].str.endswith('GCAGGT', na=False)] = np.nan

            # The last 5 bases can't be GAGAC
            df.loc[df[column].str.endswith('GAGAC', na=False)] = np.nan

            # The last 5 bases can't be CTCGA
            df.loc[df[column].str.endswith('CTCGA', na=False)] = np.nan

            # Removing NA values from dataset
            df[column].replace('nan',np.nan, inplace=True)
            df.dropna(inplace=True)

            # Removing duplicate observations (removing a duplicated row)
            df.drop_duplicates(inplace=True)

            # Removing duplicate sequences (if a sequence is repeated when
            # the Gene symbol is different,
            # then we remove all the observations with that seq value)
            df = df.drop_duplicates(subset=['sequence'], keep=False)
```

```
    return df


#============================================================
# Frequency count
#============================================================

def dataframe_to_dictionary(df, col1,col2):
    '''
    Transform a two-columns dataframe into a dictionary,
    where the first column is the key
    and the second column is the value as a list.
    If you have duplicated entries and do not want to lose them,
    then you need a list as a dictionary value
    input= dataframe
    output= dictionary
    '''
    from collections import defaultdict
    my_dict = defaultdict(list)
    for k, v in zip(df[col1].values,df[col2].values):
        my_dict[k].append(v)

    return my_dict


def top_n(d, n):
    '''function to return top n key-values pair in a dictionary'''
    dct = defaultdict(list)
    for k, v in d.items():
        dct[v].append(k)
    return sorted(dct.items())[-n:][::-1]


def get_list_count(list_of_elems):
    ''' Function that accepts a list, gets the frequency count of elements and
    returns a dictionary of elements counts in that list.
    '''
    dict_of_elems = dict()
    # Iterate over each element in list
    for elem in list_of_elems:
        # If element exists in dict then increment its value else add it in dict
        if elem in dict_of_elems:
            dict_of_elems[elem] += 1
        else:
            dict_of_elems[elem] = 1

    # Filter key-value pairs in dictionary.
```

```
    dict_of_elems = { key:value for key, value in dict_of_elems.items()  }
    # Returns a dict of elements and their frequency count
    return dict_of_elems


def get_duplicate_count(list_of_elems):
    ''' Get frequency count of duplicate elements in the given list
    if it repeated more than once '''
    dict_of_elems = dict()
    # Iterate over each element in list
    for elem in list_of_elems:
        # If element exists in dict then increment its value else add it in dict
        if elem in dict_of_elems:
            dict_of_elems[elem] += 1
        else:
            dict_of_elems[elem] = 1

    # Filter key-value pairs in dictionary. Keep pairs whose value is greater␣
 ↪than 1
    # i.e. only duplicate elements from list.
    dict_of_elems = { key:value for key, value in dict_of_elems.items() if␣
 ↪value > 1}
    # Returns a dict of duplicate elements and thier frequency count
    return dict_of_elems


def duplicate_dict(my_dict):
    '''Function takes a dictionary with values as lists of elements
    and return a new dictionary with the values as a dictionary with the␣
 ↪duplicate count.
    '''
    #if want dictionary with all the counts just run this first loop
    for key in my_dict.keys():
        my_dict[key] = get_duplicate_count(my_dict[key])
    dict_duplicates ={}
    for key, value in my_dict.items():
        if value != {}:
            dict_duplicates[key] = value
            #print(key , ' :: ', value)
    return dict_duplicates


#=============================================================
# Merging dictionaries
#=============================================================
```

```
def mergeDict(dict1, dict2):
    ''' Merge dictionaries and keep values of common keys in list,
        (first list will be from dict2 (value))'''

    dict3 = {**dict1, **dict2}
    for key, value in dict3.items():
        if key in dict1 and key in dict2:
            #dict3[key] = value + dict1[key]
            dict3[key] = merge_list(list1=value,list2=dict1[key], n=6)
    return dict3


def merge_list(list1,list2, n):
    '''
    Merge lists and keep unique values for second list.
    A maximum lenght can be given for the newlist.
    '''
    new_list = list1*1 # make a clone
    for i in range(0, len(new_list)):
        if len(new_list) < n:
            for i in range(0, len(list2)):
                if list2[i] not in new_list and len(new_list) < n:
                    new_list.append(list2[i])
        else:
            new_list[:n]
    return new_list



#=============================================================
# Join files
#=============================================================

def join_dataframes(file1, file2):
    '''
    Joining two dataframes with same number/name of columns.
    Concatenating the new observations under the originals.
    '''
    joined_file = file1.append(file2)
    return joined_file


#=============================================================
# Retriving Files
#=============================================================

def output_file(output_file, output_path, output_file_name):
```

```
    output_file.to_csv(os.path.join(output_path, output_file_name)) #, sep='\t'
```

0.4 Runing Main Function

- User input information manually
- Computation and outputs generated

[689]:
```
## Reading files

path_1 = 'broad.csv'
broad = read_files(path_1)
initial_length_1 = len(broad)
display(broad.head(6))
print('Initial_lengt Broad: ', initial_length_1)

path_2 = 'gecko.csv'
gecko = read_files(path_2)
initial_length_2 = len(gecko)
display(gecko.head(6))
print('Initial_lengt Gecko: ',initial_length_2)
```

```
  gene_symbol              sequence
0         Pzp  TTACCTCAATATAAACGACA
1         Pzp  ATGTCTGCTATGATAACTGG
2         Pzp  CACTCTGGTCTACTGCAGTG
3         Pzp  GAACTTCCTATAACTGCTTG
4       Aanat  CGGATCTCATCCAAGTAGAG
5       Aanat  CACCGCCAGTACGTGCAGGT
```

```
Initial_lengt Broad:  78637
```

```
     gene_symbol              sequence
0  0610007P14Rik  TGTCTAAGGTTTCTCAATCC
1  0610007P14Rik  CAGTGAATGGCCTCCAAGCC
2  0610007P14Rik  GGTGCTTACTTGCTACCATC
3  0610009B22Rik  ACCTCGTCGACGAAAACATG
4  0610009B22Rik  GTTCATAGCTCATGCTGCTC
5  0610009B22Rik  GAGCAGCATGAGCTATGAAC
```

```
Initial_lengt Gecko:  130209
```

[690]:
```
## Preprocessing

broad = removing_characters(broad)
final_length_1 = len(broad)
```

```
print ('rows_dropped in broad dataset: ', initial_length_1 - final_length_1)
print('Final_length Broad: ', final_length_1,'\n')

gecko = removing_characters(gecko)
final_length_2 = len(gecko)
print ('rows_dropped in gecko dataset : ', initial_length_2 - final_length_2)
print('Final_length Gecko: ', final_length_2)

# Checking for duplicated values
print('duplicate sequences broad: ', broad.duplicated('sequence', keep=False).
 ↪sum(axis = 0))
print('duplicate sequences gecko: ', gecko.duplicated('sequence', keep=False).
 ↪sum(axis = 0))

# output clean files
output_file(broad, 'output', 'clean_broad.csv')
output_file(gecko, 'output', 'clean_gecko.csv')
```

```
sequence
rows_dropped in broad dataset:  1136
Final_length Broad:  77501

sequence
rows_dropped in gecko dataset :  9139
Final_length Gecko:  121070
duplicate sequences broad:  0
duplicate sequences gecko:  0
```

[691]:
```
## Join files
joined_datasets = join_dataframes(broad, gecko)
display(joined_datasets.head(2))

# output clean files
output_file(joined_datasets, 'output', 'joined_datasets.csv')

print('lenght joined_file: ', len(joined_datasets))
```

```
  gene_symbol              sequence
0         Pzp  TTACCTCAATATAAACGACA
1         Pzp  ATGTCTGCTATGATAACTGG


lenght joined_file:  198571
```

0.4.1 2. COUNT UNIQUE SEQUENCES

[701]:
```
# transforming dataset into a dictionary
joined_datasets_dict = dataframe_to_dictionary(joined_datasets,␣
 ↪'gene_symbol','sequence')
print('joined dataset dictionary has a lenght of: ',len(joined_datasets_dict))
```

```
joined dataset dictionary has a lenght of:  22809
```

[702]:
```
joined_datasets_dict['Pzp']
```

[702]:
```
['TTACCTCAATATAAACGACA',
 'ATGTCTGCTATGATAACTGG',
 'CACTCTGGTCTACTGCAGTG',
 'GAACTTCCTATAACTGCTTG',
 'AACGAAGCTCCTCACAGACC',
 'AGACTCACCGTTTCATTCAA',
 'GAAATTCCGAGTTGTTTCTG',
 'TTCTTTATGAAGCGCTGCGT',
 'CCGCAGACAATATGTGGTGC',
 'CTTACCTCAATATAAACGAC']
```

[703]:
```
# dictionary with only duplicate values
dict_joined_duplicates = duplicate_dict(joined_datasets_dict)
```

[655]:
```
print('Total gene symbol with duplicate values: ',  len(dict_joined_duplicates))
```

```
Total gene symbol with duplicate values:  7147
```

[667]:
```
# transforming braod and gecko into dictionaries
broad_dict = dataframe_to_dictionary(broad, 'gene_symbol','sequence')
gecko_dict = dataframe_to_dictionary(gecko, 'gene_symbol','sequence')

# merge both dictinaries with a maximum lenght of 6 starting with braod dataset.
dict_lenght_6 = mergeDict(broad_dict, gecko_dict)

# transforming dictionary into a dataframe
dict_lenght_6_df = pd.DataFrame(dict_lenght_6.items(),  columns = columns_names)
display(dict_lenght_6_df.head(2))
```

	gene_symbol	sequence
0	Pzp	[AACGAAGCTCCTCACAGACC, AGACTCACCGTTTCATTCAA, G...
1	Aanat	[CCAGTGCGTTTGAGATTGAG, CCCTCTACTTGGATGAGATC, C...

[668]:
```
gene_symbol = pd.Series(dict_lenght_6_df['gene_symbol'].values)
```

[669]:
```
dict_lenght_6_df = pd.DataFrame(dict_lenght_6_df['sequence'].dropna().values.
↪tolist(),
                                ␣
↪columns=['seq1','seq2','seq3','seq4','seq5','seq6'])
```

[666]:
```
dict_lenght_6_df_2['gene_symbol'] = gene_symbol
```

[670]:
```
dict_lenght_6_df.insert(loc=0, column='gene_symbol', value = gene_symbol)
```

[671]:
```
dict_lenght_6_df.head()
```

[671]:

	gene_symbol	seq1	seq2	\
0	Pzp	AACGAAGCTCCTCACAGACC	AGACTCACCGTTTCATTCAA	
1	Aanat	CCAGTGCGTTTGAGATTGAG	CCCTCTACTTGGATGAGATC	
2	Aatk	TGCCGCCCTTCTTACAACAC	GAGGTACACTCGGGCGTCAG	
3	Abca1	AGCCTGCTGCAGGCGAATGT	GACCAACATTCGCCTGCAGC	
4	Abca4	GGTGTCCATGAACTGCGACA	TTTCCAGATTCGCTTTGTAG	

	seq3	seq4	seq5	\
0	GAAATTCCGAGTTGTTTCTG	TTCTTTATGAAGCGCTGCGT	CCGCAGACAATATGTGGTGC	
1	CCTTCATCATTGGCTCGCTG	GATGCCACGCCTTCCTGCGC	CCGCTCAATCTCAAACGCAC	
2	AGATTGGCCACGGCTGGTTT	CGAGTATGTGGCCGACTTCT	CACCACTCACCGGAGCGACC	
3	AATAAAGCCATGCCGTCTGC	CCTCGCCGGGAGTTGGATAA	CTCACACTCATGTTGTTCGT	
4	GATGGGCAACCGAGTCAGAC	CCGCACCTTGTCGCAGTTCA	CTTCAGCAGGACTGTTACCG	

	seq6
0	CTTACCTCAATATAAACGAC
1	TGTGCACCGCCAGTACGTGC
2	GCACAGCCTCCTGTACTTAA
3	TGATCTGCCGTAACATTCTC
4	CTACCTGCTCCACACGAACT

[673]:
```
# output files
output_file(dict_lenght_6_df , 'output', 'final_df.csv')
```

[ ]:
```

```