README

March 12, 2018

In [ ]:
```
# ChIP-Seq peak-caller output

In this project I determine the optimal parameters for running a ChIP-Seq peak-caller.

This file creates binary heatmaps of optimal parameter combinations
for running MACS2, a ChIP-Seq peak-caller. ChIP-Seq (Chromatin
immunoprecipitation sequencing) peak-callers identify
locations of protein-binding in DNA. MACS2 is an alogrithm for doing that.
The optimal parameter combination was defined as the combination
which maximizes the Matthews Correlation Coefficient of the data.

The input files are f_ca and f_sample, which have the optimal peak-caller
combinations for each celltype-antibody combiation and for each sample,
respectively. The other files are all of the output files that use different
```

In [ ]:
```
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib as mpl
import pandas as pd
import numpy as np
import seaborn as sns

sns.set()

f_ca = "/data/Lei_student/Hussain/ML/dm6/peakerror/optimal/optimal_ca.csv"
f_sample = "/data/Lei_student/Hussain/ML/dm6/peakerror/optimal/optimal_sample.csv"
o_ca = "/data/Lei_student/Hussain/ML/dm6/peakerror/binary_heatmap/binary_heatmap_ca.png"
o_ca_clust = "/data/Lei_student/Hussain/ML/dm6/peakerror/binary_heatmap/binary_clusterma
o_sample = "/data/Lei_student/Hussain/ML/dm6/peakerror/binary_heatmap/binary_heatmap_sam
o_sample_clust = "/data/Lei_student/Hussain/ML/dm6/peakerror/binary_heatmap/binary_clust
o_ca_sum = "/data/Lei_student/Hussain/ML/dm6/peakerror/binary_heatmap/binary_heatmap_ca_
o_sample_sum = "/data/Lei_student/Hussain/ML/dm6/peakerror/binary_heatmap/binary_heatmap
o_ca_clust_complete = "/data/Lei_student/Hussain/ML/dm6/peakerror/binary_heatmap/binary_
o_sample_clust_complete = "/data/Lei_student/Hussain/ML/dm6/peakerror/binary_heatmap/bin
o_ca_clust_single = "/data/Lei_student/Hussain/ML/dm6/peakerror/binary_heatmap/binary_cl
o_sample_clust_single = "/data/Lei_student/Hussain/ML/dm6/peakerror/binary_heatmap/binar
o_ca_clust_ward = "/data/Lei_student/Hussain/ML/dm6/peakerror/binary_heatmap/binary_clus
o_sample_clust_ward = "/data/Lei_student/Hussain/ML/dm6/peakerror/binary_heatmap/binary_
```

```
# These lines read in the input and convert them to DataFrames.
df_ca = pd.DataFrame.from_csv(f_ca)
df_sample = pd.DataFrame.from_csv(f_sample)

# These are the MACS2 peak-caller parameter combinations.
q_value = [.03, .04, .05, .06, .07]
slocal = [500, 1000, 1500]
llocal = [5000, 10000, 15000]
highmfold = [30, 40, 50, 60, 70]
lowmfold = [3, 4, 5, 6, 7]

x_ca = [] # List of dictionaries with binary heatmap values for each celltype-antibody c
x_sample = [] # The list of dictionaries for each sample.

x_ca_labels = []

for index, row in df_ca.iterrows():
    """
    Iterate through each optimal celltype-antibody combination value and add a 1.0 for t
    """
    temp_dict = {}
    temp_dict["ca"] = row["celltype"] + "_" + row["antibody"]
    temp_dict["q_value_" + str(row["q_value"])] = 1.0
    temp_dict["slocal_" + str(row["slocal"])] = 1.0
    temp_dict["llocal_" + str(row["llocal"])] = 1.0
    temp_dict["highmfold_" + str(row["highmfold"])] = 1.0
    temp_dict["lowmfold_" + str(row["lowmfold"])] = 1.0
    x_ca.append(temp_dict)

for index, row in df_sample.iterrows():
    """
    Iterate through each optimal sample combination value and add a 1.0 for the optimal
    """
    temp_dict = {}
    temp_dict["samples"] = row["sample"]
    temp_dict["q_value_" + str(row["q_value"])] = 1.0
    temp_dict["slocal_" + str(row["slocal"])] = 1.0
    temp_dict["llocal_" + str(row["llocal"])] = 1.0
    temp_dict["highmfold_" + str(row["highmfold"])] = 1.0
    temp_dict["lowmfold_" + str(row["lowmfold"])] = 1.0
    x_sample.append(temp_dict)

x_ca_df = pd.DataFrame(x_ca) # Convert the lists to DataFrames and add zero values
x_sample_df = pd.DataFrame(x_sample)

x_ca_df.index = x_ca_df["ca"] # Index the DataFrames by their labels
x_sample_df.index = x_sample_df["samples"]
```

```
x_ca_df = x_ca_df.sort_values(x_ca_df.columns.tolist()).fillna(0.0) # Sort them by colum
x_sample_df = x_sample_df.sort_values(x_sample_df.columns.tolist()).fillna(0.0)

x_ca_df2 = x_ca_df.drop(["ca"], axis=1) # Remove the first axis
x_sample_df2 = x_sample_df.drop(["samples"], axis=1)
```

In [ ]:
```
# The follow codes creates heatmaps and clustermaps of the dataframes but use the sum of

# Several of the clustermaps differ in the methods they use for clustering.
# In this code, I'm using "average", "complete", "single", and "ward."
# The mathematics behind each clustering method is found here: https://docs.scipy.org/do
# Hamming distance is used for the metric as that corresponds to binary distances.

fig = plt.figure(figsize = (14,10)) # Heatmap of optimal values for each celltype-antibo
ax = sns.heatmap(x_ca_df2, cmap=plt.cm.binary, xticklabels=True)
ax.set_yticklabels(x_ca_df.ca.values, rotation=0)
plt.savefig(o_ca)

fig = plt.figure(figsize = (14,10)) # Heatmap of optimal values for each sample
ax = sns.heatmap(x_sample_df2, cmap=plt.cm.binary, xticklabels=True)
ax.set_yticklabels(x_sample_df.samples.values, rotation=0)
plt.savefig(o_sample)

fig = plt.figure(figsize = (14,10)) # Clustermap of optimal values for each celltype-ant
ax = sns.clustermap(x_ca_df2, metric="hamming", method="average", cmap=plt.cm.binary, xt
plt.setp(ax.ax_heatmap.yaxis.get_majorticklabels(), rotation=0)
plt.savefig(o_ca_clust)

fig = plt.figure(figsize = (14,10)) # Clustermap of optimal values for each sample using
ax = sns.clustermap(x_sample_df2, metric="hamming", method="average", cmap=plt.cm.binary
plt.setp(ax.ax_heatmap.yaxis.get_majorticklabels(), rotation=0)
plt.savefig(o_sample_clust)

fig = plt.figure(figsize = (14,10)) # Clustermap of each celltype-antibody combination u
ax = sns.clustermap(x_ca_df2, metric="hamming", method="complete", cmap=plt.cm.binary, x
plt.setp(ax.ax_heatmap.yaxis.get_majorticklabels(), rotation=0)
plt.savefig(o_ca_clust_complete)

fig = plt.figure(figsize = (14,10)) # Clustermap of each sample using the "complete" met
ax = sns.clustermap(x_sample_df2, metric="hamming", method="complete", cmap=plt.cm.binar
plt.setp(ax.ax_heatmap.yaxis.get_majorticklabels(), rotation=0)
plt.savefig(o_sample_clust_complete)

fig = plt.figure(figsize = (14,10)) # Clustermap of each celltype-antibody combination u
ax = sns.clustermap(x_ca_df2, metric="hamming", method="single", cmap=plt.cm.binary, xti
plt.setp(ax.ax_heatmap.yaxis.get_majorticklabels(), rotation=0)
plt.savefig(o_ca_clust_single)
```

```
fig = plt.figure(figsize = (14,10)) # Clustermap of each sample using the "single" metho
ax = sns.clustermap(x_sample_df2, metric="hamming", method="single", cmap=plt.cm.binary,
plt.setp(ax.ax_heatmap.yaxis.get_majorticklabels(), rotation=0)
plt.savefig(o_sample_clust_single)

fig = plt.figure(figsize = (14,10)) # Clustermap of each cell-type antibody combination
ax = sns.clustermap(x_ca_df2, metric="euclidean", method="ward", cmap=plt.cm.binary, xti
plt.setp(ax.ax_heatmap.yaxis.get_majorticklabels(), rotation=0)
plt.savefig(o_ca_clust_ward)

fig = plt.figure(figsize = (14,10)) # Clustermap of each sample using the "Ward" method.
ax = sns.clustermap(x_sample_df2, metric="euclidean", method="ward", cmap=plt.cm.binary,
plt.setp(ax.ax_heatmap.yaxis.get_majorticklabels(), rotation=0)
plt.savefig(o_sample_clust_ward)

x_ca_df2_indices = x_ca_df2.sum().sort_values().index.tolist() # Sort the indices by sum
x_sample_df2_indices = x_sample_df2.sum().sort_values().index.tolist()

fig = plt.figure(figsize = (14,10))
ax = sns.heatmap(x_ca_df2[x_ca_df2_indices], cmap=plt.cm.binary, xticklabels=True)
ax.set_yticklabels(x_ca_df.ca.values, rotation=0)
plt.savefig(o_ca_sum)

fig = plt.figure(figsize = (14,10))
ax = sns.heatmap(x_sample_df2[x_sample_df2_indices], cmap=plt.cm.binary, xticklabels=Tru
ax.set_yticklabels(x_sample_df.samples.values, rotation=0)
plt.savefig(o_sample_sum)
```