Untitled

February 18, 2020

1 Final Notebook

[459]:
```
%load_ext line_profiler
```

```
The line_profiler extension is already loaded. To reload it, use:
  %reload_ext line_profiler
```

[2]:
```
from Bio.SubsMat import MatrixInfo as matlist

class SeqData:

    CONV = {
        'NUM': [0, 0, 0, 1, 1, 1, 1, 1, 2, 2, 2, 2, 3, 4, 5, 5, 6, 6, 6, 7],
        'EIIP' : [0.0946, 0.0516, 0.0548, 0.0373, 0.0057, 0.0, 0.0, 0.0823,\
                  0.0829, 0.0941, 0.0036, 0.0761, 0.0198, 0.005, 0.1263, 0.0058,␣
 ↪\
                  0.0371, 0.0242, 0.0959, 0.0829],
        'FNS' : ['Aromatic', 'Aromatic', 'Aromatic', 'Hydrophobic',␣
 ↪'Hydrophobic', \
                 'Hydrophobic', 'Hydrophobic', 'Hydrophobic', 'Polar', 'Polar',␣
 ↪'Polar', \
                 'Polar', 'Proline', 'Glycine', 'Charge (-)', 'Charge (-)',␣
 ↪'Charge (+)', \
                 'Charge (+)', 'Charge (+)', 'Excluded'],
    }
    # @TODO: Import Biopython dicts of matrices, not read from .csvs
    MATR = {
        'PAM30': pd.read_csv('./src_data/pam30.csv', index_col=0).to_dict(),
        'BLOSUM': pd.read_csv('./src_data/BLOSUM.csv', index_col=0).to_dict(),
    }
    DIST = {
        'jaro_winkler': (lambda p1, p2: td.jaro_winkler.
 ↪normalized_similarity(p1, p2)),
        'needleman_wunsch': (lambda p1, p2: td.needleman_wunsch.
 ↪normalized_similarity(p1, p2)),
        'smith_waterman': (lambda p1, p2: td.smith_waterman.
 ↪normalized_similarity(p1, p2)),
```

```
        'levenshtein': (lambda p1, p2: td.levenshtein.normalized_similarity(p1,␣
↪p2))
    }

    def __init__(self, conv=CONV.keys(), matr=MATR.keys(), dist=DIST.keys()):

        self.conv = {cnv:self.CONV[cnv] for cnv in self.CONV.keys() if cnv in␣
↪conv}
        self.matr = {mtr:self.MATR[mtr] for mtr in self.MATR.keys() if mtr in␣
↪matr}
        self.dist = {dst:self.DIST[dst] for dst in self.DIST.keys() if dst in␣
↪dist}
```

[118]:
```
'''
Class to calculate several simialrity metrics for an input list of peptides␣
↪given a sequence string to compare it to.
USAGE:
1. Create SequenceSimilarityObject({sequence string}, {dictionary of data␣
↪paths} (to be deprecated soon -- use
   internal class data), {path to peptides csv}, {column title of sequence␣
↪values of aforementioned peptides csv})
   for any number of sequences you want to be compared to the list of peptides
2. For each object, call object.generate_similarity() to fill out the Dataframe␣
↪with similarity metrics
3. To whittle down this similarity matrix to list only those peptides with␣
↪pattern matching of a minimum length
   at a matching index in the rerence peptide (henceforth the binder), call␣
↪object.get_df_with_binder_subseqs(min_length={#})
@ Author: Chris Pecunies, with help from Savvy Gupta and Aaron Tsang
@ Date: February 12, 2020
'''
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from typing import Set, Tuple, Dict, List
from scipy import stats, signal, fft
#import sci-kit learn
import textdistance as td
from Bio import pairwise2
from Bio.SubsMat import MatrixInfo as matlist

class SequenceSimilarity:
    '''
    Class that takes in a path to a list of amino acid sequences as well
    as any number of peptide sequences explicitly that are known to have
```

```
    a certain set of properties. Generates metrics for similarity for each
    peptide in path and returns domains AA sequence with high similarity
    '''
    AA = list('FYWAVILMSTNQPGDEKHRC')

    def __init__(self, binder: Tuple[str, str],
                 data: SeqData,
                 p_path: str,
                 aa_col: str,
                 dists: List = [],   #set dists = None for no dists
                 only_match: bool = False,
                 min_length: int = 0):

        # ---- setting data ---------
        self.d = {d: vars(data)[d] for d in list(vars(data).keys())}
        self.conv = list(self.d['conv'].keys())
        self.aa_map = {conv:dict(zip(self.AA, self.d['conv'][conv])) for conv␣
↪in self.conv}
        self.conv_pep = lambda t, p: [self.aa_map[t][AA] if AA in self.AA else␣
↪0 for AA in p]
        self.sim_sum = lambda p1, p2, m: sum([self.d['matr'][m][a1][a2] for a1,␣
↪a2 in zip(p1, p2)])

        self.seq = aa_col
        self.sim_cols = ['PAM30', 'BLOSUM45', 'RRM_SN', 'RRM_Corr',␣
↪'weighted_matches']
        if dists is not None:
            self.sim_cols += self.d['dist'].keys() if not dists else dists
        self.conv_cols = [conv+'_Seq' for conv in self.conv]
        self.cols = [aa_col] + self.sim_cols + self.conv_cols
        self.bname, self.b = binder
        self.bsseq = [(self.b[i:j], i) for i in range(len(self.b)) for j in␣
↪range(i+1, len(self.b)+1)]

        # ---- helper lambda functions

        self.p_og = pd.read_csv(p_path)
        self.p_og.columns = [aa_col]
        self.p_og[aa_col].drop_duplicates()
        self.p_og = self.p_og[~self.p_og[aa_col].str.contains("O")]
        self.p_sl = self.p_og[self.p_og[aa_col].str.len() == len(self.b)]
        self.p_ls = self.p_sl[aa_col].tolist()
        self.p_og = self.p_og[aa_col].tolist()
        if len(self.p_ls) == 0:
            raise Exception("No peptides of same length as binder found")
        self.p = pd.DataFrame(columns=self.cols)
        self.p[self.seq] = self.p_ls
```

```
        # @TODO P STILL HAS DUPLICATES ????
        self.update_similarities(dists)

        self.p_match = self.filter_by_bsseq(min_length)
        self.p_match_ls = self.p_match[self.seq].tolist()
        if only_match:
            self.p, self.p_ls = self.p_match, self.p_match_ls

    #----------------SET UP FUNCTIONS (void)------------------------------#
↪

    def _update_RRM_similarity(self, match_len = True) -> None:
        """
        Uses the Resonant Recognition Model as described by Irena Cosic to
        """
        rrm = pd.DataFrame(index=self.p.index)
        eiip_seq = self.p['EIIP_Seq'].tolist()
        rrm = rrm.assign(
            seq = self.p_ls,
            eiip = eiip_seq,
            dft = [np.fft.rfft(eiip)/len(eiip) for eiip in eiip_seq],
            dft2 = [2*np.abs(np.fft.rfft(eiip)/len(eiip)) for eiip in eiip_seq],
            freq = [np.abs(np.fft.fftfreq(len(eiip))[0:int(len(eiip)/2+1)]) for
↪eiip in eiip_seq],
            power = [np.abs(np.fft.rfft(eiip)/len(eiip))**2 for eiip in
↪eiip_seq],
        )
        bind_eiip = self.conv_pep('EIIP', self.b)
        self.bind_eiip_dft = 2*np.abs(np.fft.rfft(bind_eiip)/len(bind_eiip))
        self.bind_freq = np.abs(np.fft.fftfreq(len(bind_eiip))[0:
↪int(len(bind_eiip)/2+1)])
        self.bind_peak = signal.find_peaks(self.bind_eiip_dft)
        peaks = [signal.find_peaks(d) for d in rrm.dft2]
        rrm['peak_loc'] = [p[0] for p in peaks]
        rrm['peak_val'] = [p[1] for p in peaks]
        rrm['peak_dist'] = [np.abs(p[0]-self.bind_peak[0]) for p in peaks]
        rrm['correlate'] = [signal.correlate(d, self.bind_eiip_dft) for d in
↪rrm.dft2]
        rrm['convolve'] = [signal.convolve(d, self.bind_eiip_dft) for d in rrm.
↪dft2]
        self.rrm = rrm

        def get_max_seqs(num = 10, typ='correlate'):
            maxes = [signal.find_peaks(rrm.iloc[n].correlate) for n in
↪range(len(rrm))]
            maxheight = [rrm[typ].iloc[i][m[0][0]] for i, m in enumerate(maxes)]
```

```
            top_idx = np.argsort(maxheight)[-num:]
            top_values = [maxheight[i] for i in top_idx]
            top_seq = [top_seq.append(rex['seq'].iloc[i]) for i in top_idx]
            top_data = self.p[self.p[self.seq].isin(top_seq)]
            return top_data

        def plot(col = 'column'): plt.plot(rrm.[col])
        def merge(): return rrm.merge(self.p, left_on=rrm.index, right_on=self.
↪p.index)


    # NOTE! Adds columns "Matching_sseqs" and "Num_matching" to output
    # Might be too unwieldy / unhelpful for output similarity data
    # if so, just comment out _update_matching_sseqs()
    def _update_matching_sseqs(self, w1: float = 1, w2: float = 1) -> None:
        """
        Returns a number as a new column representing the number of "matches" a␣
↪peptide
        has for all possible subsequences for the binder inputted at a given␣
↪index. For
        weighting=1, all matches are treated equally ('Y' at position 3 is␣
↪treated equal
        to IMV at position 0) but lowering weighting lowers smaller-length␣
↪matches
        """
        # @TODO Remove "duplicates" which occur at different matching indexes␣
↪of binder
        # but are part of a larger pattern already recorded at an earlier index
        self.p.sseq_matches, self.p.weighted_matches = None, None
        self.bsseq.sort(key = lambda ss: len(ss[0]), reverse=True)
        score: float = lambda s: (w1 * 1) + (len(s)**w2)
        matches: List = list(); nmatches = list()
        for i, seq in enumerate(self.p_ls):
            matches.append(list()); nmatches.append(int())
            trigger = False
            for j in range(len(seq)):
                if trigger: break
                trigger = False
                for (sseq, bin_i) in self.bsseq:
                    in_seq = seq[j:len(sseq)+j]
                    if (bin_i == j) and (in_seq == sseq):
                        if len(matches[i]) > 0 and matches[i][-1][0].
↪find(sseq)>=0:
                            trigger = True
                            break
                        matches[i].append((sseq, bin_i))
```

```
                        nmatches[i] += score(sseq)
                        break
#         matches = [pairwise2.align.localxx(s, self.b) for s in self.p_ls]
        self.p.sseq_matches, self.p.weighted_matches = matches, nmatches

    def update_similarities(self, metrics: List = []) -> None:
        '''
        Updates the similarity values whenever called (for now should be only␣
↪once right
        after creating the object, ecept possibly if the Binding peptide is␣
↪updated
        (should be handled automatically)
        '''
        matrices = list(self.d['matr'].keys())
        cdata = [[self.conv_pep(c, p) for c in self.conv] for p in self.p_ls]
        mdata = [[self.sim_sum(p, self.b, m) for m in matrices] for p in self.
↪p_ls]
        self.p[self.conv_cols] = cdata
        self.p[matrices] = mdata
        self._update_matching_sseqs()
        self._update_RRM_similarity()
        if metrics is not None:
            dists = metrics if not len(metrics) == 0 else list(self.d['dist'].
↪keys())
            self.p[dists] = [[self.d['dist'][d](p, self.b) for d in dists] for␣
↪p in self.p_ls]
        # OPTIONAL
        # self._unpack_num_encoding()



    #----------MAIN CLASS FUNCTIONS (returns data) ------------------------#

    def filter_by_sseq(self, sseq: str, ind: int):
        return self.p[self.p[self.seq].str.find(sseq) == ind]

    def filter_by_bsseq(self, min_len: int = 0) -> pd.DataFrame:
        bsseq_dfs = [self.filter_by_sseq(ss,i) for (ss,i) in self.bsseq if␣
↪len(ss)>=min_len]
        return pd.concat(bsseq_dfs)

    def get_distances(self, seqs1: list, seqs2: list) -> List[float]:
        pass

    def merge_data(self, other, sep_cols = False) -> pd.DataFrame:
```

```
        # !!! IMPORTANT: "other" must also be SequenceSimilarity object␣
↪(couldnt compile)
        """
        Returns a merged Dataframe of self.p and another SequenceSimilarity's␣
↪pep_data.
        If sep_cols=True, then the other SequenceSimilarity's columns will␣
↪simply be appended
        to the returned DataFrame (self.p is unchanged). If False, results will␣
↪be averaged.
        @NOTE: This is a super naive implementatoin -- expand this to make it␣
↪more configurable
        @TODO: Take in *others as a list of arbitrarily many other␣
↪SequenceSimilarities to compare
        """
        # must be same length binders -> so same peptides of interest
        this_data = self.p.copy()
        other_data = other.p.copy()
        sim_cols = self.sim_cols.copy()
        print(sim_cols)
        if sep_cols:
            suf = ("_"+self.bname, "_"+other.bname)
            out = this_data.merge(right=other_data, on=self.seq, suffixes=(suf))
            out = out.drop_duplicates()
            return out

        new_cols = ['{}_{}_{}'.format(col, self.bname, other.bname) for col in␣
↪sim_cols]
        out_data = pd.DataFrame(index=this_data.index, columns=seq_cols +␣
↪new_cols)
        out_data[seq_cols] = this_data[seq_cols]
        both = pd.concat([this_data[non_seq_cols],other_data[non_seq_cols]])
        out_data[new_cols] = both.groupby(both.index).mean()
        if 'sseq_matches' in self.cols or 'sseq_matches' in other.cols:
            both_match = self.p['sseq_matches'].append(other.
↪pdata['sseq_matches'])
            both['sseq_matches'] = both_match
            out_data.join(both_match)
        return out_data

        #@TODO Finish


    #------------------miscellaneous methods-----------------------------#

#     def get_kendalltau_corr_map(self) -> Tuple:
```

```
#         return stats.kendalltau(self.data['AA_MAP'][['Num']], self.
 ↪data['AA_MAP'][['EIIP']])
```

[119]:
```
DATA = SeqData()
SEQS = [
    ('GRBP5','IMVTESSDYSSY'),
    ('M6','IMVTASSAYDDY')
]
AA_COL = 'Sequences'
PEP_PATH = './src_data/Sequence_data.csv'
dat1 = {
    'grbp5_sim' : SequenceSimilarity(SEQS[0], DATA, PEP_PATH, AA_COL),
    'm6_sim' : SequenceSimilarity(SEQS[1], DATA, PEP_PATH, AA_COL)
}
'''
dat2 = {
    'both_sep_sim' : dat1['grbp5_sim'].merge_data(other=dat1['m6_sim'],␣
 ↪sep_cols=True),
    'both_avg_sim' : dat1['grbp5_sim'].merge_data(other=dat1['m6_sim'],␣
 ↪sep_cols=False),
    'both_sep_sim_match' : dat1['grbp5_sim_match'].
 ↪merge_data(other=dat1['m6_sim_match'], sep_cols=True),
    'both_avg_sim_match' : dat1['grbp5_sim_match'].
 ↪merge_data(other=dat1['m6_sim_match'], sep_cols=False),
}
'''
```

```
/usr/lib/python3.7/site-packages/ipykernel_launcher.py:150: FutureWarning:
Passing list-likes to .loc or [] with any missing label will raise
KeyError in the future, you can use .reindex() as an alternative.

See the documentation here:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#deprecate-
loc-reindex-listlike
```

[119]:
```
"\ndat2 = {\n    'both_sep_sim' :
dat1['grbp5_sim'].merge_data(other=dat1['m6_sim'], sep_cols=True),\n
'both_avg_sim' : dat1['grbp5_sim'].merge_data(other=dat1['m6_sim'],
sep_cols=False),\n    'both_sep_sim_match' :
dat1['grbp5_sim_match'].merge_data(other=dat1['m6_sim_match'], sep_cols=True),\n
'both_avg_sim_match' :
dat1['grbp5_sim_match'].merge_data(other=dat1['m6_sim_match'],
sep_cols=False),\n}\n"
```

[123]:
```
d1 = list(dat1.values())
d1d = [d.p for d in d1]
rex = d1[0].rrm
```

```
rex.head()
```

[123]:

```
            seq                                                eiip  \
0  SVPHFSDEDKDP  [0.0829, 0.0057, 0.0198, 0.0242, 0.0946, 0.082…
1  VPHFSDEDKDPE  [0.0057, 0.0198, 0.0242, 0.0946, 0.0829, 0.126…
2  SVPHFSEEEKEA  [0.0829, 0.0057, 0.0198, 0.0242, 0.0946, 0.082…
3  VPHFSEEEKEAE  [0.0057, 0.0198, 0.0242, 0.0946, 0.0829, 0.005…
4  SVPHFSDEDKDP  [0.0829, 0.0057, 0.0198, 0.0242, 0.0946, 0.082…

                                                 dft  \
0  [(0.06264166666666665+0j), (-0.011294400459931…
1  [(0.05621666666666665+0j), (-0.01955731388825…
2  [(0.033975+0j), (1.021992058759744e-05-0.00823…
3  [(0.027549999999999998+0j), (-0.00143547036225…
4  [(0.06264166666666665+0j), (-0.011294400459931…

                                                dft2  \
0  [0.1252833333333333, 0.028179612473691253, 0.0…
1  [0.11243333333333333, 0.039239587699259636, 0…
2  [0.06795, 0.016479581260746716, 0.026961520934…
3  [0.055099999999999996, 0.020884774229360296, 0…
4  [0.1252833333333333, 0.028179612473691253, 0.0…

                                                freq  \
0  [0.0, 0.08333333333333333, 0.16666666666666666…
1  [0.0, 0.08333333333333333, 0.16666666666666666…
2  [0.0, 0.08333333333333333, 0.16666666666666666…
3  [0.0, 0.08333333333333333, 0.16666666666666666…
4  [0.0, 0.08333333333333333, 0.16666666666666666…

                                               power peak_loc peak_val  \
0  [0.003923978402777776, 0.0001985226397918539, …       []       {}
1  [0.003160313611111111, 0.000384936310701972, 0…       []       {}
2  [0.001154300625, 6.789414963238858e-05, 0.0001…   [3, 5]       {}
3  [0.0007590024999999999, 0.00010904344865283799…   [2, 5]       {}
4  [0.003923978402777776, 0.0001985226397918539, …       []       {}

  peak_dist                                          correlate  \
0        []  [0.006080417777777776, 0.00369355025376957, 0…
1        []  [0.005456764444444444, 0.003991765965944491, 0…
2    [0, 2]  [0.00329784, 0.0020613087035645777, 0.00266369…
3    [1, 2]  [0.002674186666666666, 0.002036545650201958, 0…
4        []  [0.006080417777777776, 0.00369355025376957, 0…

                                             convolve
0  [0.015639536111111105, 0.007005268555061297, 0…
1  [0.014035427777777776, 0.008028216530542623, 0…
```

```
2  [0.008482424999999998, 0.003948725996030621, 0…
3  [0.006878316666666666, 0.004140935319768755, 0…
4  [0.015639536111111105, 0.007005268555061297, 0…
```

[208]:
```
rex.head()
```

[208]:
```
            seq                                                eiip  \
0  SVPHFSDEDKDP  [0.0829, 0.0057, 0.0198, 0.0242, 0.0946, 0.082…
1  VPHFSDEDKDPE  [0.0057, 0.0198, 0.0242, 0.0946, 0.0829, 0.126…
2  SVPHFSEEEKEA  [0.0829, 0.0057, 0.0198, 0.0242, 0.0946, 0.082…
3  VPHFSEEEKEAE  [0.0057, 0.0198, 0.0242, 0.0946, 0.0829, 0.005…
4  SVPHFSDEDKDP  [0.0829, 0.0057, 0.0198, 0.0242, 0.0946, 0.082…

                                                 dft  \
0  [(0.06264166666666665+0j), (-0.011294400459931…
1  [(0.05621666666666665+0j), (-0.01955731388825…
2  [(0.033975+0j), (1.021992058759744e-05-0.00823…
3  [(0.027549999999999998+0j), (-0.00143547036225…
4  [(0.06264166666666665+0j), (-0.011294400459931…

                                                dft2  \
0  [0.1252833333333333, 0.028179612473691253, 0.0…
1  [0.11243333333333333, 0.039239587699259636, 0…
2  [0.06795, 0.016479581260746716, 0.026961520934…
3  [0.055099999999999996, 0.020884774229360296, 0…
4  [0.1252833333333333, 0.028179612473691253, 0.0…

                                                freq  \
0  [0.0, 0.08333333333333333, 0.16666666666666666…
1  [0.0, 0.08333333333333333, 0.16666666666666666…
2  [0.0, 0.08333333333333333, 0.16666666666666666…
3  [0.0, 0.08333333333333333, 0.16666666666666666…
4  [0.0, 0.08333333333333333, 0.16666666666666666…

                                               power peak_loc peak_val  \
0  [0.003923978402777776, 0.0001985226397918539, …       []       {}
1  [0.003160313611111111, 0.000384936310701972, 0…       []       {}
2  [0.001154300625, 6.789414963238858e-05, 0.0001…   [3, 5]       {}
3  [0.0007590024999999999, 0.00010904344865283799…   [2, 5]       {}
4  [0.003923978402777776, 0.0001985226397918539, …       []       {}

  peak_dist                                          correlate  \
0        []  [0.006080417777777776, 0.00369355025376957, 0…
1        []  [0.005456764444444444, 0.003991765965944491, 0…
2    [0, 2]  [0.00329784, 0.0020613087035645777, 0.00266369…
3    [1, 2]  [0.002674186666666666, 0.002036545650201958, 0…
4        []  [0.006080417777777776, 0.00369355025376957, 0…
```

```
                                            convolve
0  [0.015639536111111105, 0.007005268555061297, 0…
1  [0.014035427777777776, 0.008028216530542623, 0…
2  [0.008482424999999998, 0.003948725996030621, 0…
3  [0.006878316666666666, 0.004140935319768755, 0…
4  [0.015639536111111105, 0.007005268555061297, 0…
```

[164]:
```
maxes = [signal.find_peaks(rex.iloc[n].correlate) for n in range(len(rex))]
print(maxes[:5])
maxheight = [rex['correlate'].iloc[i][m[0][0]] for i, m in enumerate(maxes)]
print(maxheight[:5])
maxval = np.amax(maxheight)
maxind = np.argmax(maxheight)
maxinds = sorted(maxheight, reverse=True)
top_idx = np.argsort(maxheight)[-10:]
top_values = [maxheight[i] for i in top_idx]
top_seq = []
for i in top_idx:
    top_seq.append(rex['seq'].iloc[i])
    print(rex['seq'].iloc[i], top_values)

plt.plot(rex.iloc[maxind].correlate)
plt.grid()
plt.show()
```

```
[(array([3, 6]), {}), (array([3, 6]), {}), (array([ 3,  6,  8, 11]), {}),
(array([ 3,  6,  8, 11]), {}), (array([3, 6]), {})]
[0.003983477738678655, 0.00398398417120515, 0.003391613716366524,
0.0026732966362789864, 0.003983477738678655]
MRQYLVLSMQSS [0.01718980621246784, 0.01718980621246784, 0.01718980621246784,
0.01729663597577382, 0.017398115560299906, 0.018957853522945458,
0.020090881499368368, 0.020412413777365555, 0.02045441339375905,
0.0219375672693727]
MRQYLVLSMQSS [0.01718980621246784, 0.01718980621246784, 0.01718980621246784,
0.01729663597577382, 0.017398115560299906, 0.018957853522945458,
0.020090881499368368, 0.020412413777365555, 0.02045441339375905,
0.0219375672693727]
MRQYLVLSMQSS [0.01718980621246784, 0.01718980621246784, 0.01718980621246784,
0.01729663597577382, 0.017398115560299906, 0.018957853522945458,
0.020090881499368368, 0.020412413777365555, 0.02045441339375905,
0.0219375672693727]
FPDFYDSGEHLS [0.01718980621246784, 0.01718980621246784, 0.01718980621246784,
0.01729663597577382, 0.017398115560299906, 0.018957853522945458,
0.020090881499368368, 0.020412413777365555, 0.02045441339375905,
0.0219375672693727]
LDGLDGSGFGFD [0.01718980621246784, 0.01718980621246784, 0.01718980621246784,
```

```
0.01729663597577382, 0.017398115560299906, 0.018957853522945458,
0.020090881499368368, 0.020412413777365555, 0.02045441339375905,
0.0219375672693727]
NLDEIDRSDFGR [0.01718980621246784, 0.01718980621246784, 0.01718980621246784,
0.01729663597577382, 0.017398115560299906, 0.018957853522945458,
0.020090881499368368, 0.020412413777365555, 0.02045441339375905,
0.0219375672693727]
GDDDDNDAMELL [0.01718980621246784, 0.01718980621246784, 0.01718980621246784,
0.01729663597577382, 0.017398115560299906, 0.018957853522945458,
0.020090881499368368, 0.020412413777365555, 0.02045441339375905,
0.0219375672693727]
LDEIDRSDFGRF [0.01718980621246784, 0.01718980621246784, 0.01718980621246784,
0.01729663597577382, 0.017398115560299906, 0.018957853522945458,
0.020090881499368368, 0.020412413777365555, 0.02045441339375905,
0.0219375672693727]
DEIDRSDFGRFV [0.01718980621246784, 0.01718980621246784, 0.01718980621246784,
0.01729663597577382, 0.017398115560299906, 0.018957853522945458,
0.020090881499368368, 0.020412413777365555, 0.02045441339375905,
0.0219375672693727]
DDDDNDAMELLQ [0.01718980621246784, 0.01718980621246784, 0.01718980621246784,
0.01729663597577382, 0.017398115560299906, 0.018957853522945458,
0.020090881499368368, 0.020412413777365555, 0.02045441339375905,
0.0219375672693727]
```

[206]:
```
out = d1[0].p[d1[0].p['Sequences'].isin(top_seq)]
#out =out['Sequences'].drop_duplicates()
out.head()
```

[206]:

	Sequences	PAM30	BLOSUM45	RRM_SN	RRM_Corr	weighted_matches	\
474	FPDFYDSGEHLS	-65	NaN	NaN	NaN	2	
1103	LDGLDGSGFGFD	-38	NaN	NaN	NaN	2	
1507	MRQYLVLSMQSS	-62	NaN	NaN	NaN	2	
1539	MRQYLVLSMQSS	-62	NaN	NaN	NaN	2	
1561	MRQYLVLSMQSS	-62	NaN	NaN	NaN	2	

	jaro_winkler	needleman_wunsch	smith_waterman	levenshtein	\
474	0.477778	0.541667	0.083333	0.083333	
1103	0.277778	0.541667	0.083333	0.083333	
1507	0.611111	0.541667	0.166667	0.166667	
1539	0.611111	0.541667	0.166667	0.166667	
1561	0.611111	0.541667	0.166667	0.166667	

	NUM_Seq	\
474	[0, 3, 5, 0, 0, 5, 2, 4, 5, 6, 1, 2]	

```
1103  [1, 5, 4, 1, 5, 4, 2, 4, 0, 4, 0, 5]
1507  [1, 6, 2, 0, 1, 1, 1, 2, 1, 2, 2, 2]
1539  [1, 6, 2, 0, 1, 1, 1, 2, 1, 2, 2, 2]
1561  [1, 6, 2, 0, 1, 1, 1, 2, 1, 2, 2, 2]
```

```
                                                 EIIP_Seq FNS_Seq
474   [0.0946, 0.0198, 0.1263, 0.0946, 0.0516, 0.126…     -13
1103  [0.0, 0.1263, 0.005, 0.0, 0.1263, 0.005, 0.082…      -2
1507  [0.0823, 0.0959, 0.0761, 0.0516, 0.0, 0.0057, …     -10
1539  [0.0823, 0.0959, 0.0761, 0.0516, 0.0, 0.0057, …     -10
1561  [0.0823, 0.0959, 0.0761, 0.0516, 0.0, 0.0057, …     -10
```

[174]:
```
print(d1[0].p.describe())
print(d1[0].p[d1[0].p['Sequences'].isin(top_seq)].describe())
```

```
       weighted_matches  jaro_winkler  needleman_wunsch  smith_waterman  \
count       3078.000000   3078.000000       3078.000000     3078.000000
mean           1.139701      0.419312          0.526492        0.064138
std            1.350734      0.107040          0.032280        0.066640
min            0.000000      0.000000          0.500000        0.000000
25%            0.000000      0.388889          0.500000        0.000000
50%            0.000000      0.444444          0.500000        0.083333
75%            2.000000      0.477778          0.541667        0.083333
max            7.000000      0.674603          0.666667        0.333333

       levenshtein
count  3078.000000
mean      0.066629
std       0.067072
min       0.000000
25%       0.000000
50%       0.083333
75%       0.083333
max       0.333333
       weighted_matches  jaro_winkler  needleman_wunsch  smith_waterman  \
count         15.000000     15.000000         15.000000       15.000000
mean           1.666667      0.496667          0.536111        0.122222
std            0.899735      0.139567          0.021517        0.061935
min            0.000000      0.277778          0.500000        0.000000
25%            2.000000      0.388889          0.541667        0.083333
50%            2.000000      0.611111          0.541667        0.166667
75%            2.000000      0.611111          0.541667        0.166667
max            3.000000      0.611111          0.583333        0.166667

       levenshtein
count    15.000000
mean      0.122222
```

```
std       0.061935
min       0.000000
25%       0.083333
50%       0.166667
75%       0.166667
max       0.166667
```

[168]:
```
top_seq[0]
```

[168]:
```
'MRQYLVLSMQSS'
```

[328]:
```
d1[2].pdata.groupby('weighted_matches').mean().plot()
```

[328]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8a414a0650>
```

[ ]:
```
self.bsseq.sort(key = lambda ss: len(ss[0]), reverse=True)
        score: float = lambda s: (single_match_weight * 1) + (len(s)**weight)
        matches: List = list(); nmatches = list()
        for i, seq in enumerate(self.p_ls):
            matches.append(list()); nmatches.append(int())
            iter_skip = 0
            for j in range(len(seq)):
                if iter_skip > 0:
                    iter_skip -= 1
                    continue
                for (sseq, bin_i) in self.bsseq:
                    in_seq = inseq
                    if (bin_i == j) and (in_seq == sseq):
                        matches[i].append((sseq, bin_i))
                        nmatches[i] += score(sseq)
                        iter_skip += len(sseq)
                        continue
        self.p.sseq_matches, self.p.weighted_matches = matches, nmatches
```

[209]:
```
merge = rex.merge(d1[0].p, left_on=rex.index, right_on=d1[0].p.index)

corr = [sum([merge.iloc[i].power for m in row]) for i, row in enumerate(merge)]
print(corr[0], len(corr), len(merge))
corr = [np.interp(corr[i], (min(corr[i]), max(corr[i])), (0,1)) for i in␣
 ↪range(len(merge))]
ds = []; c = []
for dist in list(d['dist'].keys()):
```

```
    distances = [np.interp(merge[dist].iloc[i], (merge[dist].iloc[i].min(),␣
 ↪merge[dist].iloc[i].max()), (0,1))]
    ds.append(distances)
    c.apppend(np.corrcoef(corr, distances))[1,2]
merge.plot(corr, ds[0])
```

```
[1.96198920e-02 9.92613199e-04 7.33654514e-04 2.34262847e-04
 3.80461806e-05 6.87574995e-04 5.57501701e-03] 25 3078
```

```
      ␣
 ↪---------------------------------------------------------------------------

       IndexError                                Traceback (most recent call␣
 ↪last)

       <ipython-input-209-44b817db2eb2> in <module>
         3 corr = [sum([merge.iloc[i].power for m in row]) for i, row in␣
 ↪enumerate(merge)]
         4 print(corr[0], len(corr), len(merge))
    ----> 5 corr = [np.interp(corr[i], (min(corr[i]), max(corr[i])), (0,1)) for␣
 ↪i in range(len(merge))]
         6 ds = []; c = []
         7 for dist in list(d['dist'].keys()):


       <ipython-input-209-44b817db2eb2> in <listcomp>(.0)
         3 corr = [sum([merge.iloc[i].power for m in row]) for i, row in␣
 ↪enumerate(merge)]
         4 print(corr[0], len(corr), len(merge))
    ----> 5 corr = [np.interp(corr[i], (min(corr[i]), max(corr[i])), (0,1)) for␣
 ↪i in range(len(merge))]
         6 ds = []; c = []
         7 for dist in list(d['dist'].keys()):


       IndexError: list index out of range
```

[205]:
```
merge
```

[205]:

	key_0	seq	eiip	\
0	0	SVPHFSDEDKDP	[0.0829, 0.0057, 0.0198, 0.0242, 0.0946, 0.082…	
1	1	VPHFSDEDKDPE	[0.0057, 0.0198, 0.0242, 0.0946, 0.0829, 0.126…	
2	2	SVPHFSEEEKEA	[0.0829, 0.0057, 0.0198, 0.0242, 0.0946, 0.082…	
3	3	VPHFSEEEKEAE	[0.0057, 0.0198, 0.0242, 0.0946, 0.0829, 0.005…	
4	4	SVPHFSDEDKDP	[0.0829, 0.0057, 0.0198, 0.0242, 0.0946, 0.082…	

```
…      …           …                                                    …
3073   3073  FLRRIRPKLKWD  [0.0946, 0.0, 0.0959, 0.0959, 0.0, 0.0959, 0.0…
3074   3074  LRRIRPKLKWDN  [0.0, 0.0959, 0.0959, 0.0, 0.0959, 0.0198, 0.0…
3075   3075  RRIRPKLKWDNQ  [0.0959, 0.0959, 0.0, 0.0959, 0.0198, 0.0371, …
3076   3076  YGGFLRRQFKVV  [0.0516, 0.005, 0.005, 0.0946, 0.0, 0.0959, 0…
3077   3077  GGFLRRQFKVVT  [0.005, 0.005, 0.0946, 0.0, 0.0959, 0.0959, 0…

                                                        dft  \
0     [(0.06264166666666665+0j), (-0.011294400459931…
1     [(0.05621666666666665+0j), (-0.01955731388825…
2     [(0.033975+0j), (1.021992058759744e-05-0.00823…
3     [(0.027549999999999998+0j), (-0.00143547036225…
4     [(0.06264166666666665+0j), (-0.011294400459931…
…                                                    …
3073  [(0.05478333333333333+0j), (0.0120289691495536…
3074  [(0.0472+0j), (0.006376852056801644-0.00215376…
3075  [(0.05354166666666666+0j), (0.0120914403741765…
3076  [(0.047266666666666665+0j), (-0.01882832480253…
3077  [(0.05080833333333333+0j), (-0.013883726825491…

                                                       dft2  \
0     [0.1252833333333333, 0.028179612473691253, 0.0…
1     [0.11243333333333333, 0.039239587699259636, 0…
2     [0.06795, 0.016479581260746716, 0.026961520934…
3     [0.055099999999999996, 0.020884774229360296, 0…
4     [0.1252833333333333, 0.028179612473691253, 0.0…
…                                                    …
3073  [0.10956666666666666, 0.026094853577604493, 0…
3074  [0.0944, 0.01346148948945863, 0.02273646728153…
3075  [0.10708333333333332, 0.025799178373241267, 0…
3076  [0.09453333333333333, 0.037744954227810754, 0…
3077  [0.10161666666666666, 0.030682013917474927, 0…

                                                       freq  \
0     [0.0, 0.08333333333333333, 0.16666666666666666…
1     [0.0, 0.08333333333333333, 0.16666666666666666…
2     [0.0, 0.08333333333333333, 0.16666666666666666…
3     [0.0, 0.08333333333333333, 0.16666666666666666…
4     [0.0, 0.08333333333333333, 0.16666666666666666…
…                                                    …
3073  [0.0, 0.08333333333333333, 0.16666666666666666…
3074  [0.0, 0.08333333333333333, 0.16666666666666666…
3075  [0.0, 0.08333333333333333, 0.16666666666666666…
3076  [0.0, 0.08333333333333333, 0.16666666666666666…
3077  [0.0, 0.08333333333333333, 0.16666666666666666…

                                                      power peak_loc peak_val  \
```

```
0     [0.003923978402777776, 0.0001985226397918539, …       []       {}
1     [0.003160313611111111, 0.000384936310701972, 0…       []       {}
2     [0.001154300625, 6.789414963238858e-05, 0.0001…   [3, 5]       {}
3     [0.0007590024999999999, 0.00010904344865283799…   [2, 5]       {}
4     [0.003923978402777776, 0.0001985226397918539, …       []       {}
…                                                    …      …      …
3073  [0.003001213611111111, 0.0001702353458091545, …      [4]       {}
3074  [0.00222784, 4.530292481870129e-05, 0.00012923…      [3]       {}
3075  [0.002866710069444444, 0.00016639940118357997,…      [3]       {}
3076  [0.0022341377777777777, 0.0003561703924148822,…      [4]       {}
3077  [0.002581486736111111, 0.00023534649450803128,…      [4]       {}

     peak_dist  … RRM_SN RRM_Corr weighted_matches jaro_winkler  \
0           []  …    NaN      NaN                2     0.477778
1           []  …    NaN      NaN                2     0.472222
2       [0, 2]  …    NaN      NaN                2     0.472222
3       [1, 2]  …    NaN      NaN                0     0.388889
4           []  …    NaN      NaN                2     0.477778
…         … …    …      …              …          …
3073       [1]  …    NaN      NaN                0     0.444444
3074       [0]  …    NaN      NaN                0     0.444444
3075       [0]  …    NaN      NaN                0     0.444444
3076       [1]  …    NaN      NaN                0     0.000000
3077       [1]  …    NaN      NaN                0     0.000000

     needleman_wunsch smith_waterman levenshtein  \
0            0.541667       0.166667    0.166667
1            0.541667       0.083333    0.083333
2            0.541667       0.083333    0.083333
3            0.500000       0.000000    0.000000
4            0.541667       0.166667    0.166667
…               …            …         …
3073         0.500000       0.000000    0.000000
3074         0.500000       0.000000    0.000000
3075         0.500000       0.000000    0.000000
3076         0.500000       0.000000    0.000000
3077         0.500000       0.000000    0.000000

                                  NUM_Seq  \
0     [2, 1, 3, 6, 0, 2, 5, 5, 5, 6, 5, 3]
1     [1, 3, 6, 0, 2, 5, 5, 5, 6, 5, 3, 5]
2     [2, 1, 3, 6, 0, 2, 5, 5, 5, 6, 5, 1]
3     [1, 3, 6, 0, 2, 5, 5, 5, 6, 5, 1, 5]
4     [2, 1, 3, 6, 0, 2, 5, 5, 5, 6, 5, 3]
…                                       …
3073  [0, 1, 6, 6, 1, 6, 3, 6, 1, 6, 0, 5]
3074  [1, 6, 6, 1, 6, 3, 6, 1, 6, 0, 5, 2]
```

```
3075  [6, 6, 1, 6, 3, 6, 1, 6, 0, 5, 2, 2]
3076  [0, 4, 4, 0, 1, 6, 6, 2, 0, 6, 1, 1]
3077  [4, 4, 0, 1, 6, 6, 2, 0, 6, 1, 1, 2]
```

		EIIP_Seq	FNS_Seq
0	[0.0829, 0.0057, 0.0198, 0.0242, 0.0946, 0.082…		-8
1	[0.0057, 0.0198, 0.0242, 0.0946, 0.0829, 0.126…		-3
2	[0.0829, 0.0057, 0.0198, 0.0242, 0.0946, 0.082…		-6
3	[0.0057, 0.0198, 0.0242, 0.0946, 0.0829, 0.005…		-5
4	[0.0829, 0.0057, 0.0198, 0.0242, 0.0946, 0.082…		-8
…	…	…	…
3073	[0.0946, 0.0, 0.0959, 0.0959, 0.0, 0.0959, 0.0…		-15
3074	[0.0, 0.0959, 0.0959, 0.0, 0.0959, 0.0198, 0.0…		-15
3075	[0.0959, 0.0959, 0.0, 0.0959, 0.0198, 0.0371, …		-4
3076	[0.0516, 0.005, 0.005, 0.0946, 0.0, 0.0959, 0…		-14
3077	[0.005, 0.005, 0.0946, 0.0, 0.0959, 0.0959, 0…		-21

[3078 rows x 25 columns]

[ ]: