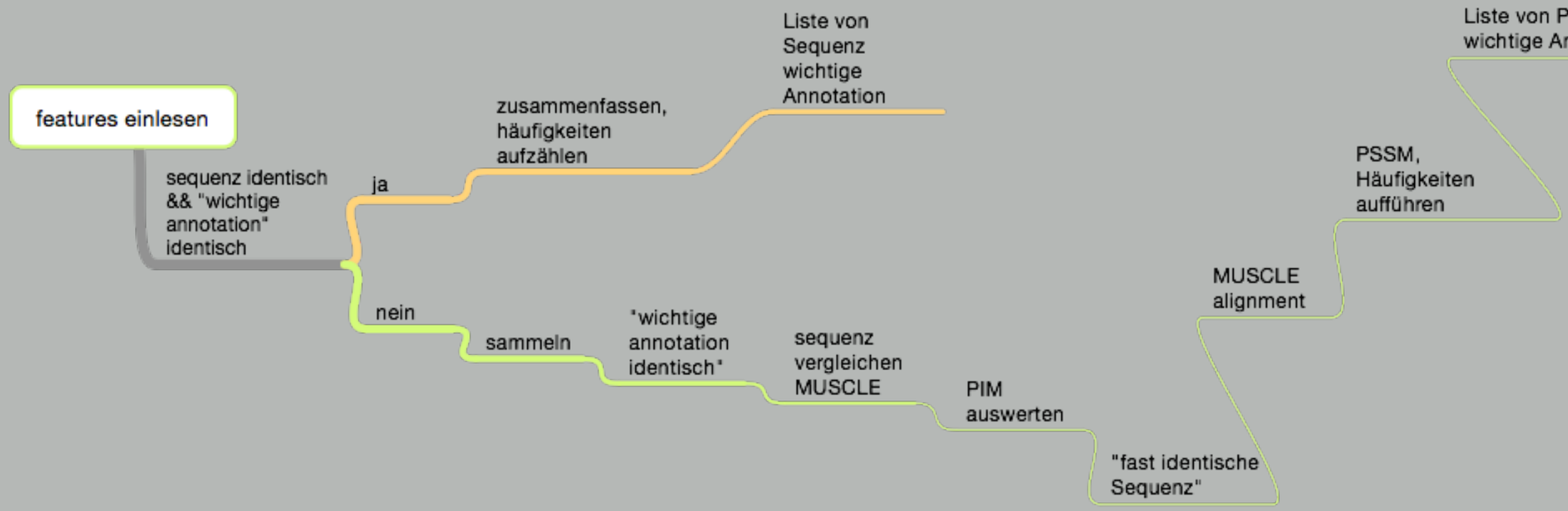
Plasmidanalyse

05.01.2015

Überblick


features einlesen
sequenz identisch && "wichtige annotation" identisch
ja
zusammenfassen, häufigkeiten aufzählen
Liste von Sequenz wichtige Annotation
nein
sammeln
'wichtige annotation identisch'
sequenz vergleichen MUSCLE
PIM auswerten
"fast identische Sequenz"
MUSCLE alignment
PSSM, Häufigkeiten aufführen
Liste von PSSM + wichtige Annotations

```
# - - - - start of skript - - - -
# - - - - - - - - - - - - - - - - -
jeremyFeatures = ['oriT', 'polyA_signal', 'rep_origin', 'primer_bind', 'rRNA', 'mRNA', 'tRNA']
dominiks_list = ['promoter', 'RBS', '-10_signal', '-35_signal']
kevins_list = ['terminator', 'CDS']
alessandros_list = ['protein_bind', 'misc_binding', 'misc_recomb', 'LTR', 'misc_signal',
                    'enhancer', 'mobile_element', 'sig_peptide']


complete_list = jeremyFeatures + dominiks_list + kevins_list + alessandros_list

save_file_object = open("list_of_identical_objects.txt", "w")

# Schwellenwert fuer nahezu identische Sequenzen bei der percent identitiy matrix
schwellenwert = 90.0

for feature in alessandros_list:
    print 'Feature: ' + feature
    filePath = "../../files/vectors.gb"
    # make a list generator with the desired feature and its annotation
    list_generator = generateList(feature, filePath)

    #  same sequences + annotations -> count occurences and prepare new list
    list_of_identical_objects = reduce_to_single_sequences(list_generator, feature)
    summe = 0
    for object in list_of_identical_objects:
        summe += object.getOccurences()
        #Blast typical sequence
        save_file_object.write(str(object) + "\t" + str(object.getOccurences()) + "\n")
    print("Anzahl identischer objekte: \t" + str(len(list_of_identical_objects)))
    print("Summe aller Objekte: \t\t\t" + str(summe))
    # 'wichtige Annotation' Sequenzen in Liste speichern und MUSCLE uebergeben
    prepared_list = group_identical_annotations(list_of_identical_objects, feature)
    #prepared_list = list_of_identical_objects
    for entry in prepared_list:
        print str(entry)
        muscle_result = clustering(entry, 1)
        # PIM Auswertung: Sequenzen groesser Schwellenwert (bsp. 95%) rausspeichern. Rueckgabe: Liste von "fast identische Sequenzen"
        list_of_near_identical_sequences = pim_evaluation(schwellenwert)
        for sequences in list_of_near_identical_sequences:
            print 'Sequences for further inspection: ' + str(sequences)
            if len(sequences) > 1:
                clustering(sequences, 2)
                pssm_result = createPSSM()
                f = open('pssm_results.txt', 'a')
                f.write(pssm_result)
                f.close()

save_file_object.close()
```

Output

Feature: oriT
Anzahl identischer objekte: 40
Summe aller Objekte: 116
MUSCLE
Status: RUNNING
Status: RUNNING
Status: RUNNING
['identifier3', 'identifier1']
['identifier11', 'identifier17']
['identifier4', 'identifier17']
['identifier4', 'identifier11']
['identifier14', 'identifier17']
['identifier14', 'identifier11']
['identifier14', 'identifier4']

Percent Identity Matrix

```
#
#
#  Percent Identity  Matrix - created by Clustal2.1
#
#
```

1:	identifier2	100.00	36.33	41.07	41.07	47.30	58.82	58.42	58.25	58.42	46.98	52.84	43.84	36.84
2:	identifier0	36.33	100.00	41.15	41.15	48.26	59.63	59.26	59.09	58.33	48.26	52.54	50.00	50.18
3:	identifier1	41.07	41.15	100.00	100.00	38.94	61.47	61.11	60.00	58.33	41.12	54.77	39.08	39.11
4:	identifier3	41.07	41.15	100.00	100.00	38.97	61.47	61.11	60.00	58.33	41.12	54.77	39.11	39.14
5:	identifier17	47.30	48.26	38.94	38.97	100.00	100.00	100.00	100.00	99.09	96.12	99.59	99.87	99.86
6:	identifier11	58.82	59.63	61.47	61.47	100.00	100.00	100.00	100.00	99.07	100.00	100.00	100.00	100.00
7:	identifier4	58.42	59.26	61.11	61.11	100.00	100.00	100.00	100.00	99.07	100.00	100.00	100.00	100.00
8:	identifier14	58.25	59.09	60.00	60.00	100.00	100.00	100.00	100.00	99.09	100.00	100.00	100.00	100.00
9:	identifier16	58.42	58.33	58.33	58.33	99.09	99.07	99.07	99.09	100.00	99.09	99.09	99.09	99.09
10:	identifier19	46.98	48.26	41.12	41.12	96.12	100.00	100.00	100.00	99.09	100.00	100.00	99.16	99.15
11:	identifier22	52.84	52.54	54.77	54.77	99.59	100.00	100.00	100.00	99.09	100.00	100.00	100.00	100.00

Sequence 1 & 3

>identifier1
ATCGATGATAAGCTGTCAAAGATGAGAATTAATTCCACGGACTATAGACTATACTAGATA
CTCCGTCTACTGTACGATACACTTCCGCTCAGGTCCTTGTCCTTTAACGAGGCCTTACCA
CTCTTTTGTTACTCTATTGATCCAGCTCAGCAAAGGCAGTGTGATCTAAGATTCTATCTT
CGCGATGTAGTAAAACTAGCTAGACCGAGAAAGAGACTAGAAATGCAAAAGGCACTTCTA
CAATGGCTGCCATCATTATTATCCGATGTGACGCTGCAGCTTCTCAATGATATTCGAATA
CGCTTTGAGGAGATACAGCCTAATATCCGACAAACTGTTTTACAGATTTACGATCGTACT
TGTTACCCATCATTGAATTTTGAACATCCGAACCTGGGAGTTTTCCCTGAAACAGATAGT
ATATTTGAACCTGTATAATAATATATAGTCTAGCGCTTTACGGAAGACAATGTATGTATT
TCGGTTCCTGGAGAAACTATTGCATCTATTGCATAGGTAATCTTGCACGTCGCATCCCCG
GTTCATTTTCTGCGTTTCCATCTTGCACTTCAATAGCATATCTTTGTTAACGAAGCATCT
GTGCTTCATTTTGTAGAACAAAAATGCAACGCGAGAGCGCTAATTTTTCAAACAAAGAAT
CTGAGCTGCATTTTTACAGAACAGAAATGCAACGCGAAAGCGCTATTTTACCAACGAAGA
ATCTGTGCTTCATTTTTGTAAAACAAAAATGCAACGCGACGAGAGCGCTAATTTTTCAAA
CAAAGAATCTGAGCTGCATTTTTACAGAACAGAAATGCAACGCGAGAGCGCTATTTTACC
AACAAAGAATCTATACTTCTTTTTTGTTCTACAAAAATGCATCCCGAGAGCGCTATTTTT
CTAACAAAGCATCTTAGATTACTTTTTTTCTCCTTTGTGCGCTCTATAATGCAGTCTCTT
GATAACTTTTTGCACTGTAGGTCCGTTAAGGTTAGAAGAAGGCTACTTTGGTGTCTATTT
TCTCTTCCATAAAAAAAGCCTGACTCCACTTCCCGCGTTTACTGATTACTAGCGAAGCTG
CGGGTGCATTTTTTCAAGATAAAGGCATCCCCGATTATATTCTATACCGATGTGGATTGC
GCATACTTTGTGAACAGAAAGTGATAGCGTTGATGATTCTTCATTGGTCAGAAAATTATG
AACGGTTTCTTCTATTTTGTCTCTATATACTACGTATAGGAAATGTTTACATTTTCGTAT
TGTTTTCGATTCACTCTATGAATAGTTCTTACTACAATTTTTTTGTCTAAAGAGTAATAC
TAGAGATAAACATAAAAAATGTAGAGGTCGAGTTTAGATGCAAGTTCAAGGAGCGAAAGG
TGGATGGGTAGGTTATATAGGGATATAGCACAGAGATATATAGCAAAGAGATACTTTTGA
GCAATGTTTGTGGAAGCGGTATTCGCAATG

>identifier3
TTATCGATGATAAGCTGTCAAAGATGAGAATTAATTCCACGGACTATAGACTATACTAGA
TACTCCGTCTACTGTACGATACACTTCCGCTCAGGTCCTTGTCCTTTAACGAGGCCTTAC
CACTCTTTTGTTACTCTATTGATCCAGCTCAGCAAAGGCAGTGTGATCTAAGATTCTATC
TTCGCGATGTAGTAAAACTAGCTAGACCGAGAAAGAGACTAGAAATGCAAAAGGCACTTC
TACAATGGCTGCCATCATTATTATCCGATGTGACGCTGCAGCTTCTCAATGATATTCGAA
TACGCTTTGAGGAGATACAGCCTAATATCCGACAAACTGTTTTACAGATTTACGATCGTA
CTTGTTACCCATCATTGAATTTTGAACATCCGAACCTGGGAGTTTTCCCTGAAACAGATA
GTATATTTGAACCTGTATAATAATATATAGTCTAGCGCTTTACGGAAGACAATGTATGTA
TTTCGGTTCCTGGAGAAACTATTGCATCTATTGCATAGGTAATCTTGCACGTCGCATCCC
CGGTTCATTTTCTGCGTTTCCATCTTGCACTTCAATAGCATATCTTTGTTAACGAAGCAT
CTGTGCTTCATTTTGTAGAACAAAAATGCAACGCGAGAGCGCTAATTTTTCAAACAAAGA
ATCTGAGCTGCATTTTTACAGAACAGAAATGCAACGCGAAAGCGCTATTTTACCAACGAA
GAATCTGTGCTTCATTTTTGTAAAACAAAAATGCAACGCGACGAGAGCGCTAATTTTTCA
AACAAAGAATCTGAGCTGCATTTTTACAGAACAGAAATGCAACGCGAGAGCGCTATTTTA
CCAACAAAGAATCTATACTTCTTTTTTGTTCTACAAAAATGCATCCCGAGAGCGCTATTT
TTCTAACAAAGCATCTTAGATTACTTTTTTTCTCCTTTGTGCGCTCTATAATGCAGTCTC
TTGATAACTTTTTGCACTGTAGGTCCGTTAAGGTTAGAAGAAGGCTACTTTGGTGTCTAT
TTTCTCTTCCATAAAAAAAGCCTGACTCCACTTCCCGCGTTTACTGATTACTAGCGAAGC
TGCGGGTGCATTTTTTCAAGATAAAGGCATCCCCGATTATATTCTATACCGATGTGGATT
GCGCATACTTTGTGAACAGAAAGTGATAGCGTTGATGATTCTTCATTGGTCAGAAAATTA
TGAACGGTTTCTTCTATTTTGTCTCTATATACTACGTATAGGAAATGTTTACATTTTCGT
ATTGTTTTCGATTCACTCTATGAATAGTTCTTACTACAATTTTTTTGTCTAAAGAGTAAT
ACTAGAGATAAACATAAAAAATGTAGAGGTCGAGTTTAGATGCAAGTTCAAGGAGCGAAA
GGTGGATGGGTAGGTTATATAGGGATATAGCACAGAGATATATAGCAAAGAGATACTTTT
GAGCAATGTTTGTGGAAGCGGTATTCGCAATG

PSSM Result

PSSM done

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
-:	0.94	0.94	0.94	0.94	0.94	0.94	0.94	0.94	0.94	0.94	0.94	0.94	0.94	0.94	0.94	0.94	0.94	0.94	0.94	0.94	0.94	0.94	0.94	0.94	0.94
A:	-1.38	0.94	-1.38	0.94	-1.38	0.94	0.94	0.94	0.94	-1.38	-1.38	-1.38	-1.38	0.94	-1.38	-1.38	0.94	0.94	-1.38	-1.38	-1.38	-1.38	-1.38	-1.38	-1.38
C:	0.94	-1.38	0.94	-1.38	-1.38	-1.38	-1.38	-1.38	-1.38	-1.38	-1.38	-1.38	-1.38	-1.38	-1.38	-1.38	0.94	-1.38	-1.38	-1.38	-1.38	-1.38	-1.38	-1.38	0.94
G:	-1.38	-1.38	-1.38	-1.38	-1.38	-1.38	-1.38	-1.38	-1.38	0.94	-1.38	0.94	-1.38	-1.38	0.94	-1.38	-1.38	-1.38	-1.38	-1.38	-1.38	-1.38	-1.38	-1.38	-1.38
T:	-1.38	-1.38	-1.38	-1.38	0.94	-1.38	-1.38	-1.38	-1.38	-1.38	0.94	-1.38	0.94	-1.38	-1.38	-1.38	-1.38	-1.38	0.94	0.94	0.94	-1.38	-1.38	-1.38	-1.38

Aufbau des Codes

ResultObject

```
class ResultObject:
    """
    An Object for storing the sequence, feature type and annotation of the feature
    """
    def __init__(self, sequence, feature_type, annotation):
        self.occurences = 0
        self.annotation = annotation
        self.feature_type = feature_type
        self.sequence = sequence
    def __str__(self):
        return str(self.feature_type)+"; " + str(self.sequence)+ "; " + str(self.annotation)

    def setOccurences(self):
        self.occurences += 1
    def getOccurences(self):
        return self.occurences
```

Generierung der einzelnen Objekte

```
def generateList(feature_type):
    """
    A generator
    :param feature_type:
    :return: a ResultObject with the desired sequence and annotation
    """
    for record in records:
        if len(record.seq) > 1500: # minimum for number of bases
            for feature in record.features:
                if feature.type == feature_type:
                    sequence_of_feature = record.seq[feature.location.start: feature.location.end]
                    annotation = feature.qualifiers
                    feature_type = feature.type
                    result = ResultObject(sequence_of_feature, feature_type, annotation)
                    yield result
```

Herausstreichen von Duplikaten

```
def reduce_to_single_sequences(generated_object, feature):
    """
    same sequences + annotations -> count occurences and prepare new list
    :param generated_object: list generator
    :return: list of identical objects
    """

    featureTypes = {
                    'oriT': ['gene', 'product'], 'polyA_signal': ['note'], 'rep_origin': ['note'],
                    'primer_bind': ['note'], 'rRNA': ['poduct'], 'mRNA': ['gene'], 'tRNA': ['product'],
                    'promoter': ['note'], "RBS": ['note', 'gene'], "-10_signal": ['note', 'gene'],
                    '-35_signal': ['note', 'gene'], 'terminator': ['note'], 'CDS': ['gene', 'product'],
                    'protein_bind': ['note', 'bound_moiety'], 'misc_binding': ['note', 'bound_moiety'],
                    'misc_recomb': ['note'], 'LTR': ['note'], 'misc_signal': ['note'], 'enhancer': ['note'],
                    'mobile_element': ['mobile_element_type', 'note'], 'sig_peptide': ['note']
                    }
    results = []
    try:
        results.append(generated_object.next())
    except (StopIteration):
        print "Warning: empty generator. ", feature, " not found!"
        return []
    #print results

    for resultObject in generated_object:
        counter = 1
        foundMatch = False
        for result in results:
            matchCounter = 0
            for key in featureTypes[feature]:
                if str(resultObject.sequence) == str(result.sequence) \
                        and resultObject.annotation.get(key)==result.annotation.get(key):
                    matchCounter += 1
            if matchCounter == len(featureTypes[feature]):
                result.setOccurences()
                foundMatch = True
            if len(results) == counter:
                if foundMatch == False:
                    results.append(resultObject)
            counter += 1

    return results
```

Sequenzen mit wichtiger Annotation speichern

```
def group_identical_annotations(single_sequence_list, feature):
    print 'Identical annotations with different Sequences are grouped'

    featureTypes = {
    'terminator':{
        'T0': 'note', 'T1': 'note', 'T2': 'note',
        'T7': 'note', 'rrnB': 'note', 'tNOS': 'note'
    },
    'CDS': {
        'hypothetical protein': 'product', 'bla': 'gene', 'ampR': 'gene',
        'kanamycin resistance protein': 'product', 'Amp': 'product', 'tetR': 'product',
        'cat': 'gene', 'green fluorescent protein': 'product', 'neo': 'gene'
    },

# and resultValue == annotationKey

    save_list = []
    print featureTypes[feature]
    counter = 0
    print feature
    for resultKey, resultValue in featureTypes[feature].items():
        counter += 1
        tempSequenceList = single_sequence_list
        tempList = []
        for resultObject in tempSequenceList:
            for annotationKey, annotationValue in resultObject.annotation.items():
                if resultKey == annotationValue[0] and resultValue == annotationKey:
                    tempList.append(resultObject)
        save_list.append(tempList)
    print "*---*"
    print len(save_list)
    print 'save list: ' + str(save_list)
    return save_list
```

MUSCLE Multialignment

```
def clustering(objects_of_sequences, durchgang):
    """
    MUSCLE
    Compare the sequences to similarity. same sequences with
similar annotations shall be clustered
    :param objects_of_sequences:
    :return:
    """
    list_of_sequences = ""

    if len(objects_of_sequences)<=1:
        return []

    if durchgang == 2:
        print 'MUSCLE 2. Durchgang'
        for (i,k) in enumerate(objects_of_sequences):
            #print k
            list_of_sequences += ">" + "identifier" + str(i)
+"\n"+str(k)+"\n\n"
    else:
        print 'MUSCLE 1. Durchgang'
        for (i,k) in enumerate(objects_of_sequences):
            #print k
            list_of_sequences += ">" + "identifier" + str(i)
+"\n"+str(k.sequence)+"\n\n"

    #print list_of_sequences
    m = MUSCLE(verbose=False)
    jobid = m.run(frmt="fasta", sequence=list_of_sequences,
email="dominik.burri1@students.fhnw.ch")

    while m.getStatus(jobid) == u'RUNNING':
        print "Status: ", m.getStatus(jobid)
```

```
if durchgang == 2:
        result=m.getResult(jobid, "aln-fasta")
        sequencelist = result
        f = open('fastatmp', 'w')
        f.write(sequencelist)
        f.close()
    else:
        resultFile = open('results.txt', 'a')
        result=m.getResult(jobid, "sequence")
        sequencelist = result
        f = open('sequence_result.fasta', 'w')
        f.write(sequencelist)
        resultFile.write(sequencelist)
        f.close()
        result=m.getResult(jobid, "pim")
        pim_result = result
        f = open('pim_result.txt', 'w')
        f.write(pim_result)
        resultFile.write(pim_result)
        f.close()
        resultFile.close()

return sequencelist
```

speichert die Werte in separate Files

PIM Auswertung

1

```python
def pim_evaluation(schwellenwert):
    '''
    Auswertung der Percent Identity Matrix
    Nimmt die bestehenden Files zur Berechnung: pim_result.txt und sequence_result.fasta
    :param schwellenwert: der Schwellenwert fuer die Erkennung von Matches
    :return: Liste mit aehnlichen Sequenzen (als Seq Object gespeichert),
    die jeweils in eine Liste gepackt sind
    '''

    identifier_list = []
    matches = []

    f = open('pim_result.txt', 'r')
    for i in range(6):
        f.readline()
    lines = 1
    while True:
        line = f.readline()
        if line == '':
            break
        words = line.split()
        words.pop(0) # deleting 1: etc
        name = words[0] # getting 'identifierXY'
        identifier_list.append(name)
        words.pop(0) # deleting 'identifierXY'
        if len(words) >= 1:
            index = 1
            for value in words:
                value = float(value)
                if index < lines:
                    if value > schwellenwert:
                        matches.append([name, index])
                        #print name, secondname
                else:
                    break
                index += 1
        lines += 1
```

PIM Auswertung 2

2

```
# get the correct index from the full identifier list
# and set the name of the corresponding identifier
names = []
new_matches = []
for match in matches:
    match[1] = identifier_list[match[1]-1]

    if match[1] in names:
        if not match[0] in new_matches:
            new_matches.append(match[0])
    else:
        names.append(match[1])
    print match # print the identifier names
    # if not match[0] in names:
    #     names.append(match[0])
    # if not match[1] in names:
    #     names.append(match[1])

# TODO: get the multiple sequences that are similar
multiple_similar_sequences = []
for new_match in new_matches:
    multiple_similar_sequences.append(new_match)
    for match in matches:
        if new_match in match:
            for entry in match:
                if not entry in multiple_similar_sequences:
                    multiple_similar_sequences.append(entry)
```

3

```
#
# print 'Multiple similar sequences: ' + str(multiple_similar_sequences)
# TODO: get the unnessecary entries out
# for match in matches:
#     print match
#     if match[0] in multiple_similar_sequences:
#         matches.remove(str(match))
#     if match[1] in multiple_similar_sequences:
#         matches.remove(str(match))

matches.append(multiple_similar_sequences)

print 'Matches: ' + str(matches)
# get the sequence from the identifier name
handle = open('sequence_result.fasta', 'r')
for record in SeqIO.parse(handle, 'fasta', IUPAC.unambiguous_dna):
    for match in matches:
        for i in range(len(match)):
            if record.id == match[i]:
                match[i] = record.seq

        #print match

handle.close()
return matches
```

Create PSSM

```
def createPSSM(sequencelist):
    print "Start PSSM"

    if len(sequencelist)==0:
        return

    #sequencelist = sequencelist.replace("-", ".")
    f = open('fastatmp', 'w')
    f.write(sequencelist)
    f.close()
```

```
    list = []

    for seq_record in SeqIO.parse("fastatmp", "fasta", IUPAC.unambiguous_dna):
        list.append(str(seq_record.seq))

    #Blast typical sequence
    result_handle = NCBIWWW.qblast("blastn", "nt", list[0])
    save_file = open("my_blast.xml", "w")
    save_file.write(result_handle.read())
    save_file.close()
    result_handle.close()

    #motifs.create(test, alphabet=Gapped(IUPAC.unambiguous_dna))
    m = motifs.create(list, alphabet=Gapped(IUPAC.unambiguous_dna))
    print "motif created"


    pwm = m.counts.normalize(pseudocounts=0.25)
    print "PWM done"
    pssm = pwm.log_odds()
    print "PSSM done"
    return pssm
```