# Hands-on Programming Session

Laboratory of Bioinformatics I (Module 2)

*Author:*
Emidio Capriotti
emidio.capriotti@unibo.it

University of Bologna (Italy)

May 2, 2020

## Contents

# 1 Protein Structure Superimposition

**Exercise 1: Write a python script for superimposing two 3D structures.**

The script that should take in input two PDB structures, the relative chains and their residue intervals. The script has to include the following steps:

1. Parse the PDB file
2. Select the subset of residues
3. Calculate the transformation that minimizues the RMSD

The last step is should be performed using the biopython library. In particular we should use the module SVDSuperimposer that allow to calculate the best transformation and the RMSD.

For addressing this task we build a basic python script with two functions.

1. Parse the PDB file, select the subset of residues
2. Call SVDSuperimposer and return the transformation (Rotation matrix and Transaltion vector) and the RMSD.

As a reference for the selection of the coordinates in the PDB file consider the following fields in the ATOM record:

- ATOM: 1-4
- ATOM TYPE: 13-16
- RESIDUE TYPE: 18-20
- CHAIN: 22
- RESIDUE NUMBER: 23-26
- X-COORD: 31-38
- Y-COORD: 39-46
- Z-COORD: 47-54

Below is reported a simple script to solve this exercise.

```python
#!/usr/bin/env python
import sys
from Bio.SVDSuperimposer import SVDSuperimposer
import numpy as np

def get_list(naa):
  vpos=[]
  l1=naa.split(',')
  for i in l1:
    l2=i.split('-')
    if len(l2)==1:
      vpos=vpos+l2
    elif len(l2)==2:
      try:
        vpos=vpos+[str(j) for j in range(int(l2[0]),int(l2[1])+1)]
      except:
        sys.stderr.write('ERROR: Incorrect input format '+naa)
        sys.exit(1)
    else:
      sys.stderr.write('ERROR: Incorrect input format '+naa)
      sys.exit(1)
  return vpos

def get_ca_atoms(pdbfile,chain,rlist,atom='CA'):
    l_coord=[]
    fpdb=open(pdbfile)
    for line in fpdb:
      if line[:4] != 'ATOM': continue
      if line[21] != chain: continue
      if line[22:26].strip() not in rlist: continue
      if line[12:16].strip() != atom: continue
      x=float(line[30:38])
      y=float(line[38:46])
      z=float(line[46:54])
      l_coord.append([x,y,z])
    return l_coord

def get_rmsd(coord1,coord2):
    if len(coord1)!=len(coord2):
        sys.stderr.write('ERROR: The sets of coordinates have different size.')
        sys.exit(1)
    svd=SVDSuperimposer()
    svd.set(np.array(coord1),np.array(coord2))
    svd.run()
    rmsd=svd.get_rms()
    rot,tran=svd.get_rotran()
    return rot,tran,rmsd

if __name__ == '__main__':
  if len(sys.argv)==7:
    pdbfile1=sys.argv[1]
    pdbfile2=sys.argv[2]
    chain1=sys.argv[3]
    chain2=sys.argv[4]
    list1=get_list(sys.argv[5])
    list2=get_list(sys.argv[6])
    l_coord1=get_ca_atoms(pdbfile1,chain1,list1)
    l_coord2=get_ca_atoms(pdbfile2,chain2,list2)
    rot,tran,rmsd=get_rmsd(l_coord1,l_coord2)
    print (f'R= [[ {rot[0][0]:7.3f}, {rot[0][1]:7.3f}, {rot[0][2]:7.3f} ],\n'+\
           f'    [ {rot[1][0]:7.3f}, {rot[1][1]:7.3f}, {rot[1][2]:7.3f} ],\n'+\
           f'    [ {rot[2][0]:7.3f}, {rot[2][1]:7.3f}, {rot[2][2]:7.3f} ]]')
    print (f'T= [ {tran[0]:5.3f}, {tran[1]:5.3f}, {tran[2]:5.3f} ]')
    print (f'RMSD= {rmsd:5.3f}')
  else:
    print ("python super_pdb.py pdb1 pdb2 chain1 chain2 residues1 residues2")
```

```
!python script/super_pdb.py data/3O20.pdb data/3ZCF.pdb A A 10-60 10-60
```

```
R= [[  -0.216,   0.360,   0.908 ],
    [  -0.845,  -0.534,   0.011 ],
    [   0.489,  -0.765,   0.420 ]]
T= [ 32.090, 7.979, 0.263 ]
```

```
RMSD= 0.331
```

**WARNING**: This code is not safe in presence of Alternate locations in column 17 of the ATOM field.

## 2 Protein Sequence Alignment Analysis

**Exercise 2: Analyze a multuple sequence alignment to detect conserved sites.**

We collected a set of 5 cytromes for different species. The proteins have the following UniProt ID

- P99999 (human)
- P00004 (horse)
- P0C0X8 (Rhodobacter)
- P00091 (Rhodopseudomonas)
- Q93VA3 (Arabidopsis)

We build a uniqeu fasta file containing all the sequences of the proteins dowloading them autocmatically from the UniProt Rest API wit the *wget* command.

```
for i in P99999 P00004 P0C0X8 P00091 Q93VA3
do
  wget https://www.uniprot.org/uniprot/$i.fasta
done

cat P99999.fasta P00004.fasta P0C0X8.fasta P00091.fasta Q93VA3.
   ↪ fasta >cytc_aln.clw
```

You can now calculate the multiple sequence alignment of the five sequences using one of the most pupular akugnment methods such as CLUSTALW, MUSCLE and TCOFFEE.

You should save the output in clustalw format that contains seperated columns representing the identifiers and the relative aligned sequences.

An example of the output is available here.

Now you should write a python script that analyzes the multiple sequence alignment and calculates for each position the most abundant residue with its frequencyy and the information entropy (S). Where S is

$$S = \sum_{i=0}^{20} -p_i log(p_i)$$

and $p$ are the frequecnies of the amino acids. The program will have a function the parse the alignment file and a function that calculates the profile in each position of the alignment.

```
%%writefile parse_aln.py
#!/usr/bin/python
import sys
import numpy as np

def get_aln(alnfile):
  d_aln={}
  f=open(alnfile,'r')
  for line in f:
    if line.find('sp')!=0: continue
    l=line.split()
    sid=l[0]
    seq=l[1]
    d_aln[sid]=d_aln.get(sid,'')+seq
  return d_aln

def get_profile(d_aln):
  profile=[]
  n=len(list(d_aln.values()))
```

```python
  sids=d_aln.keys()
  for i in range(n):
    aas=[d_aln[j][i] for j in sids]
    vaas=get_iprofile(aas)
    tot=float(vaas[:20].sum())
    vaas[:20]=vaas[:20]/tot
    profile.append(vaas)
  return profile

def get_iprofile(aas,aa_list='ACDEFGHIKLMNPQRSTVWY-'):
  v=np.zeros(len(aa_list))
  for aa in aas:
    pos=aa_list.find(aa)
    if pos>-1: v[pos]=v[pos]+1
  return v

def print_profile(profile,aa_list='ACDEFGHIKLMNPQRSTVWY-'):
  n=len(profile)
  for i in range(n):
    pi=profile[i][:20]
    s=0.0
    for j in range(20):
      if pi[j]>0: s=s-pi[j]*np.log(pi[j])
    pm=pi.argmax()
    print (i+1,aa_list[pm],s,pi[pm],profile[i][20])

if __name__ == '__main__':
  alnfile=sys.argv[1]
  d_aln=get_aln(alnfile)
  profile=get_profile(d_aln)
  print_profile(profile)
```

```
!python parse_aln.py ../data/cytc_aln.clw
```

```
1 M 0.0 1.0 4.0
2 R 0.0 1.0 4.0
3 L 0.0 1.0 4.0
4 V 0.0 1.0 4.0
5 L 0.0 1.0 4.0
```

The output of the program will return for each position the most frequent amino acid, the entropy, the frequency and the number of gaps.

Sorting this file you can detect the most conserved sites among the five proteins.

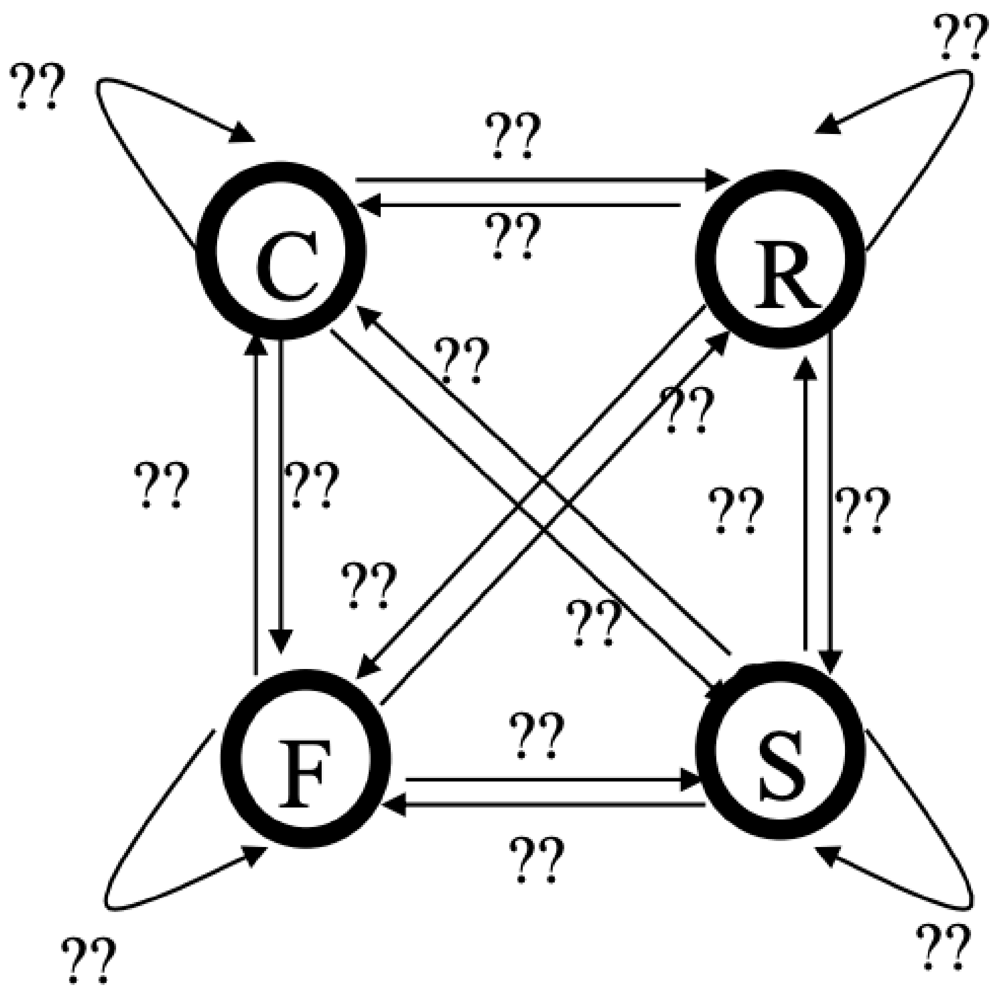

*Figure 3.1:* Probabistic Model

## 3 Probabilistic Models

**Exercise 3: Write a python script to maximize the probability of a given sequence.**

Given the probabilistic model represented with the graph below, calculate the transition probabilities that maximize the probability of the following sequence of events

CCCFFCRRRCCSSSSFSFRRFFSSF

The python script takes in input a string representing a sequence of events and returns the transition matrix and the probability of the input sequence.

```python
#!/usr/bin/python
import sys, random
import numpy as np

def get_alphabet(seq):
  alpha=[]
  for a in seq:
      if a not in alpha: alpha.append(a)
  alpha.sort()
  return ''.join(alpha)

def get_tmatrix(seq,alpha):
  n=len(alpha)
  tm=np.zeros((n,n))
  l=len(seq)
  for i in range(l-1):
    p1=alpha.find(seq[i])
    p2=alpha.find(seq[i+1])
    if p1>-1 and p2>-1: tm[p1][p2]+=1
  for i in range(n):
    tm[i,:]=tm[i,:]/np.sum(tm[i,:])
  return tm

def get_prob(seq,tm,alpha):
  p=1.0
  l=len(seq)
  for i in range(l-1):
    p1=alpha.find(seq[i])
    p2=alpha.find(seq[i+1])
    if p1>-1 and p2>-1: p=p*tm[p1][p2]
  return p

def print_logp(p):
  if p>0.:
    lp=-np.log10(p)
  else:
    lp=np.inf
  return lp

def get_shuffle(seq):
  l=[i for i in seq]
  random.shuffle(l)
  return ''.join(l)

def print_matrix(m,alpha):
  print ('Transition Matrix:')
  n=len(alpha)
  for i in range(n):
    for j in range(n):
      print (alpha[i],'->',alpha[j],'%.3f\t' %m[i][j], end = '')
    print ('')

if __name__ == '__main__':
  if len(sys.argv)<2:
    print ('python markov_model.py string_of_events')
    sys.exit()
  seq=sys.argv[1]
  alpha=get_alphabet(seq)
  tm=get_tmatrix(seq,alpha)
  p=get_prob(seq,tm,alpha)
  print_matrix(tm,alpha)
  print ("\n-Log(Probability):",'%.3f' %print_logp(p))
```

The script from the impot sequence generates the alphabet with the function *get_alphabet*.

The *get_tmatrix* funtion starting from position 0 and ending at the position n-1 counts all the transitions bewteen state i and i+1. In the final step of the function the matrix is normalized by line.

The normalization corrseponds to set to 1 the sum of probabilities of trantion from one state.

The function *get_shuffle* is written to generate random shuffled sequence and show that their probabilities are in general lower than the probability of the input sequence.

```
!python markov_model.py CCCFFCRRRCCSSSSFSFRRFFSSF
```

```
Transition Matrix:
C -> C 0.500 C -> F 0.167 C -> R 0.167 C -> S 0.167
F -> C 0.167 F -> F 0.333 F -> R 0.167 F -> S 0.333
R -> C 0.200 R -> F 0.200 R -> R 0.600 R -> S 0.000
S -> C 0.000 S -> F 0.429 S -> R 0.000 S -> S 0.571-

Log(Probability): 10.842
```

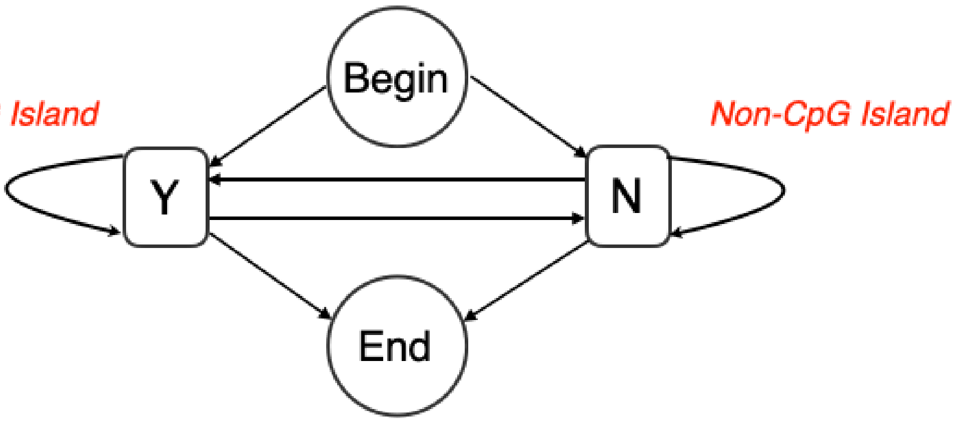

*Figure 4.1:* CpG-Island

## 4 CpG Island Hidden Markov Model

**Exercise 4: Write a script for generating and CpG Island HMM.**

For this exercise we consider as a training sequence the human chromosome 21 downloaded from the ucsc genome browser. For the GpC Island annotation refer to the cpgIslandExt.txt.gz file.

From the cpgIslandExt.txt file extract the information relative to chromosome 21.

For modeling generation of CpG Island in nucleoride sequences we use the following Hidden Markpv Model wgere the states Y and N can emit the letters representing the 4 nucleotides.

The script takes in input bedfile extracted from the cpgIslandExt.txt and the sequence of the chromosome 21 and calculates the transition and emission probabilities for each state.

```python
#!/usr/bin/env python
import sys
import numpy as np
import pickle

def get_seq(seqfile):
  seq=''
  for line in open(seqfile):
    if line[0]!='>':seq=seq+line.rstrip()
  return seq

def get_ranges(bedfile):
  l=[]
  for line in open(bedfile):
    v=line.split()
    l.append([v[1],int(v[2]),int(v[3])])
  return l

def match_seq(bed,seq):
  s=''
  n=len(bed)
  p0=0
  for i in range(n):
    s=s+(bed[i][1]-p0-1)*'N'
```

```python
      s=s+(bed[i][2]-bed[i][1]+1)*'Y'
      p0=bed[i][2]
    s=s+(len(seq)-bed[i][2])*'N'
    return s

def count(seq,cpg,nuc='ACGT',state='NY'):
    t=np.zeros((len(state),len(state)))
    e=np.zeros((len(state),len(nuc)))
    n=len(seq)
    for i in range(1,n):
      ps=state.find(cpg[i])
      pn=nuc.find(seq[i])
      pf=state.find(cpg[i-1])
      if ps>-1 and pn>-1:
        e[ps][pn]+=1
        if pf>-1: t[ps][pf]+=1
    e[0]=e[0]/np.sum(e[0])
    e[1]=e[1]/np.sum(e[1])
    t[0]=t[0]/np.sum(t[0])
    t[1]=t[1]/np.sum(t[1])
    return e,t

def print_results(e,t,nuc='ACGT',state='NY'):
    for i in range(len(state)):
      for j in range(len(state)):
        print (state[i],'->',state[j],'%.6e\t' %t[i][j],end='')
      print ('')
    for i in range(len(state)):
      print ('State:',state[i],end='\t')
      for j in range(len(nuc)):
          print (nuc[j]+':','%.3f\t' %e[i][j],end='')
      print ('')

if __name__ == '__main__':
    if len(sys.argv)<3:
      print ('python cpg-hmm.py bedfile seqfile')
      sys.exit()
    hmm={}
    state='NY'
    nuc='ACGT'
    bedfile=sys.argv[1]
    seqfile=sys.argv[2]
    seq=get_seq(seqfile)
    bed=get_ranges(bedfile)
    cgp=match_seq(bed,seq)
    seq=seq.upper()
    e,t=count(seq,cgp)
    print_results(e,t)
    hmm['E']=e
    hmm['T']=t
    pickle.dump(hmm,open('../data/cpg-hmm.pik','wb'))
```

The first 3 functions of the scripr parse the bedfile and sequfile and generated 2 strings containing the DNA sequence and the CpG state. Only standard nucleic adids are considered for the statistics.

This calculation is performed on the first 50 lines of the cpgIslandExt.txt file and the first 10 million nucleotides of chromosome 21. The pickle library is used to save the resulting HMM transission end emidiio probabilities on a file ('../data/cpg-hmm.pik') for future calculations.

```
!python cpg-hmm.py ../data/chr21_short.cpg ../data/chr21_short.fa
```

```
N -> N 9.999863e-01 N -> Y 1.370383e-05
Y -> N 1.208167e-03 Y -> Y 9.987918e-01
State: N  A: 0.280  C: 0.216  G: 0.211  T: 0.293
State: Y  A: 0.132  C: 0.357  G: 0.342  T: 0.170
```

We can now write a simple script that calculates the probability of the sequence giving the model using the forward algorithm.

```python
#!/usr/bin/env python
import sys
import numpy as np
import pickle

def calculate_forward(seq,t,e,pb,pe,nuc='ACGT',state='NY'):
  n=len(seq)
  ns=len(state)
  m=np.zeros((ns,n))
  fp=0.0
  for i in range(ns):
    m[i][0]=pb[i]*e[i][nuc.find(seq[0])]
  for i in range(1,n):
    for j in range(ns):
      for k in range(ns):
        m[j][i]=m[j][i]+m[k][i-1]*t[k][j]*e[j][nuc.find(seq[i])]
  for i in range(ns):
    fp=fp+m[i][-1]*pe[i]
  return m,fp

def calculate_logforward(seq,t,e,pb,pe,nuc='ACGT',state='NY'):
  n=len(seq)
  ns=len(state)
  m=np.zeros((ns,n))
  fp=0.0
  for i in range(ns):
    m[i][0]=-np.log10(pb[i]*e[i][nuc.find(seq[0])])
  for i in range(1,n):
    for j in range(ns):
      tmp=0.0
      for k in range(ns):
        tmp=tmp+10**-(m[k][i-1]-m[0][i-1])*t[k][j]*e[j][nuc.find(seq
    ↪ [i])]
      m[j][i]=m[0][i-1]-np.log10(tmp)
  tmp=0.0
  for i in range(ns):
    tmp=tmp+10**-(m[i][-1]-m[0][-1])*pe[i]
  fp=m[0][-1]-np.log10(tmp)
  return m,fp

if __name__ == '__main__':
  if len(sys.argv)<3:
    print ('python forward-hmm.py seqfile picklefile')
    sys.exit()
  state='NY'
  nuc='ACGT'
  seq=sys.argv[1]
  picklefile=sys.argv[2]
  hmm=pickle.load(open(picklefile,'rb'))
  # Is assumed that pbegin is 0.5
  pb=np.array([0.5,0.5])
  # Is assumed that pend is 0.5 - thiis not correct
  pe=np.array([0.5,0.5])
  t=hmm['T']
  e=hmm['E']
  m,fp=calculate_logforward(seq,t,e,pb,pe)
  print ('-Log10 Probability:','%.3f' %fp)
```

```
!python forward-hmm.py ACGTG ../data/cpg-hmm.pik
```

```
-
Log10 Probability: 3.366
```

# 5 BLAST-based Annotation

**Exercise 5: Build a BLAST-based method for protein annotation.**

In preparation of the project we will build a BLAST-based method for the annotation of BPTI/Kinutz domain. For this purpose we collect a positive set of proteins containing the BPTI/Kinutz domain from UniProt and a negative set from the same database. Divide the human and non-human proteins to build a training and testing sets respectively.

**Selecting training set from UniProt:**

- Positives: database:(type:pfam pf00014) organism:"Homo sapiens (Human) [9606]" AND reviewed:y
- Negatives: NOT database:(type:pfam pf00014) AND reviewed:yes AND organism:"Homo sapiens (Human) [9606]"

**Selecting testing set from UniProt:**

- Positives: database:(type:pfam pf00014) NOT organism:"Homo sapiens (Human) [9606] AND reviewed:y
- Negatives: NOT database:(type:pfam pf00014) AND reviewed:yes NOT organism:"Homo sapiens (Human) [9606]"

The composition of the sets will be the following:

- Human BPTI/Kunitz: 18
- BPTI/Kunitz Non Human: 18
- Not BPTI/Kunitz Human: 20347

For the negatives we divide the set in two (Training and Testing) selecting the first 10000 proteins as training and the renaining ones as Testing.

In my file I see that protein O95081 is the 10000th one and that sequence finishat line 109074, thus we split the file using the following command.

```
head -n 109074 Human_NotPF00014.fasta  >Human_NotPF00014_Training.
    ↪ fasta
tail -n +109075 Human_NotPF00014.fasta  >Human_NotPF00014_Testing.
    ↪ fasta
```

For performing the BLAST serch we need to format out positive training set of human BPTI/Kinutz proteins.

```
formatdb -i Human_PF00014.fasta
```

```
grep ">" Human_NotPF00014.fasta |head -n 10000 |tail -n 1
```

Run BLAST on the Human_PF00014.fasta dataset as a consistency test. We weant to check that all the human sequence are

```
# For checking purposes
blastpgp -i Human_PF00014.fasta -d Human_PF00014.fasta -o
    ↪ Human_PF00014.bl8 -m 8

# sort based on the 11 column to check the highst e-value
sort -grk 11 Human_PF00014.bl8| head -n 1
```

**WARNING**: we should use the option -g for sorting instead of -n. This could be different depending on linux ditribution.

We now work on the negative Training set

```
# Run on negatives
blastpgp -i Human_NotPF00014_Training.fasta -d Human_PF00014.fasta -
    o Human_NotPF00014_Training.bl8 -m 8

# sort the output to chekc the sequence with lowest e-value
sort -gk 11 Human_NotPF00014_Training.bl8 |head -n 2 |awk '{print $1
    ↪ ,$2,$11}'

sp|A6NMZ7|CO6A6_HUMAN sp|P12111|CO6A3_HUMAN 0.0
sp|A8TX70|CO6A5_HUMAN sp|P12111|CO6A3_HUMAN 0.0
```

We observe that there are many non BPTI/Kunitz proteins tha get a low e-value. Moving to the training set we can now run BLAST on both positive and negative sets.

```
# Run on positive testing set
blastpgp -i NotHuman_PF00014.fasta -d Human_PF00014.fasta -o
    ↪ NotHuman_PF00014.bl8 -m 8

# Run on negative testing set
blastpgp -i Human_NotPF00014_Testing.fasta -d Human_PF00014.fasta -o
    ↪  Human_NotPF00014_Testing.bl8 -m 8
```

We need first to rank the output on the posotive set of non Human BPTI/Kunitz proteins. To do so we grep based on the identifiers and select the mathch with lowest e-value.

```
for i in `awk '{print $1}' NotHuman_PF00014.bl8 |sort -u `
do
  grep $i NotHuman_PF00014.bl8 |sort -gk 11 |head -n 1
done > NotHuman_PF00014.bl8.best
```

Now let's select an e-value of 0.001 as a classification threshold for the classification of BPTI/kunitz proteins. Accorddin to this assumption we calculated the number of proteins with maximu e-value below that threshold.

```
# Test on positives
awk '{if ($11<0.001) print $1}'  NotHuman_PF00014.bl8.best  |sort -u
    ↪  |wc -l
  340

awk '{if ($11<0.001) print $1}'  Human_NotPF00014_Training.bl8  |
    ↪ sort -u |wc
  290

awk '{if ($11<0.001) print $1}'  Human_NotPF00014_Tresting.bl8  |
    ↪ sort -u |wc -l
  200
```

According to this results and considering the non Human BPTI/Kunits as a positive in training and testing we caclulate the following confusion matrices

| M-Training | Real_Positive | Real_Negative |
|---|---|---|
| Pred_Positive | 340 | 290 |
| Pred_Negative | 1 | 9710 |

| M-Testing | Real_Positive | Real_Negative |
|---|---|---|
| Pred_Positive | 340 | 200 |
| Pred_Negative | 1 | 10147 |

We can now calculate the performace of the method selecting a threshold of 0.001 in terms of accuracy and Matthews correlation coefficient.

```python
import math

# Training confusion matrix
mtrain=[[340,290],[1,9710]]

# Training confusion matrix
mtest=[[340,200],[1,10147]]

def accuracy(m):
  return float(m[0][0]+m[1][1])/(sum(m[0])+sum(m[1]))

def mcc(m):
  d=(m[0][0]+m[1][0])*(m[0][0]+m[0][1])*(m[1][1]+m[1][0])*(m[1][1]+
   ↪ m[0][1])
  return (m[0][0]*m[1][1]-m[0][1]*m[1][0])/math.sqrt(d)

print ('ACC Training= %5.3f' %accuracy(mtrain))
print ('MCC Testing= %5.3f\n' %mcc(mtrain))

print ('ACC Training= %5.3f' %accuracy(mtest))
print ('MCC Testing= %5.3f\n' %mcc(mtest))
```

```
ACC Training= 0.972
MCC Testing= 0.723

ACC Training= 0.981
MCC Testing= 0.785
```

We can observe that the performance in treiing and testing are different especially in terms of Matthews correlation coefficient.

To soleve the saame problem with more genetral solution we can write a python script that analyze a file derived from the BLAST output.

The program will take in input a file with three columns representing the following data

1. Protein ID
2. The lowest e-value associated to each protein
3. The class (0: NotBPTI/Kunitz 1: BPTI/Kunitz)

```python
#!/usr/bin/python
import sys, math

```

```python
def get_blast(filename):
  flist=[]
  d={}
  f=open(filename)
  for line in f:
    v=line.rstrip().split()
    d[v[0]]=d.get(v[0],[])
    d[v[0]].append([float(v[1]),int(v[2])])
  for v in d.values():
    v.sort()
    flist.append(v[0])
  return flist

def get_cm(data,th):
  # CM=[[TP,FP],[FN,TN]]
  # 0 = Negatives 1=Positives
  cm=[[0.0,0.0],[0.0,0.0]]
  for i in data:
    if i[0]<th and i[1]==1:
      cm[0][0]=cm[0][0]+1
    if i[0]>=th and i[1]==1:
      cm[1][0]=cm[1][0]+1
    if i[0]<th and i[1]==0:
      cm[0][1]=cm[0][1]+1
    if i[0]>th and i[1]==0:
      cm[1][1]=cm[1][1]+1
  return cm

def get_acc(cm):
  return float(cm[0][0]+cm[1][1])/(sum(cm[0])+sum(cm[1]))

def mcc(m):
  d=(m[0][0]+m[1][0])*(m[0][0]+m[0][1])*(m[1][1]+m[1][0])*(m[1][1]+
   ↪ m[0][1])
  return (m[0][0]*m[1][1]-m[0][1]*m[1][0])/math.sqrt(d)

if __name__ == "__main__":
  filename=sys.argv[1]
  #th=float(sys.argv[2])
  data=get_blast(filename)
  for i in range(20):
    th=10**-i
    cm=get_cm(data,th)
    print 'TH:',th,'ACC:',get_acc(cm),'MCC:',mcc(cm),cm
```

To run this progra we need to generate the input file form the the BLAST output extracting the protein ID from the 1st column and the e-value from the 11th column and adding 0 or 1 depending on the protein files we are considering.

In our case we should run the following commands to extract the data from the 3 files we have:

```
# Command for generating the positive set
awk '{split($1,v,'|'); print v[2],$11,1}' NotHuman_PF00014.bl8 >
 ↪ positive.txt

# Command for generating the negative Training set
```

```
awk '{split($1,v,'|'); print v[2],$11,0}' Human_NotPF00014_Training
    ↪ .bl8 > negative_train.txt

# Command for generating the negative Testing set
awk '{split($1,v,'|'); print v[2],$11,0}' Human_NotPF00014_Testing.
    ↪ bl8 > negative_test.txt
```

Now you can merge positive and negatives in a unique file with the cat command and give the resulting file in input to the python script above.

```
cat positive.txt negative_train.txt > train_data.txt
python performance.py train_data.txt

cat positive.txt negative_test.txt > test_data.txt
python performance.py test_data.txt
```

**WARNING**: Remember that BLAST do not return any output for sequence that match with an e-value higher tha 10 for this reason the total number of seuences you will have in the pervious file will not sum up to the total number of proteins in yor set.

You can use the *comm* commad to determine the missing sequences and run again the script over the corrected files.

# 6 Project Workflow

**Main steps for the implementation of the project.**

**Main Aim**: Developemnet of a method for the detection BPTI/Kunitz domain in proteins.

**Methodology**: Hidden Markov Model based on structural alignment.

The workflow of the project is summarized as follows

1. Selection of a representative set of protein structures from PDB.
2. Multiple structural alignment with web available tools.
3. Generation of the Hidden Markow Model for modeling BPTI/Kunitz domain.
4. Selection of training and testing set from UniProt.
5. Method optimization and assessment.

**Selection a representative set of protein structures from PDB.**

For this task we use the advanced search on the PDB web page. It is possible to use different contrains in the search. For the selection of the structures it is important to consider:

- The resolution
- The length of the protein
- External identifiers for BPTI/Kunitz domain (e.g. PFAM, SCOP CATH)

When a list of PDB strutures is returned if it is feasible you should look to the proteins to check if they have a BPTI/Kunitz domain. From the PDB web page it is possible to download the using the sequence report it is possible to download and the protein sequence and chec for the possibl redundance in the set of proteins. To check for the redundancy in the dataset you can use *blastclast* algorithm. Blastclust takes in imput a fasta faile containing a set of sequences and cluster them according to the level of sequnece identity and coverage between the proteins. The option *-S* and *-L* for the program are used to control the percentage of sequence identity and the coverage respectively.

```
#The command sed and awk can be used to clean the csv file returned
 ↪ by the PDB.

# Replace inner comma
sed 's/, /_/g' filename

# Remove spaces
sed 's/ //g' filename

# split by comma
awk -F "," '{print $1,$2}'  filename

# After genereting a fasta file run blustclust with sequen identity
 ↪ of 95% and 90% coverage.
blastclust -i seqfile.fasta -o seqfile.clust -S 95 -L 0.9
```

When the cluster file is returned select a representative structure for each cluster considering the resolution of the structure and the length of the protein.

**Multiple structural alignment with web available tools.**

After collecting a clean set of protein structures calculate the multiple structure alignment using web available tools such as PDBeFold, PROMALS3D mTM-align. Check the returned alignment and look for the RMSD between pair of structures and the sequence identity. This information can be important to detect possible errors in the initial selection of the BPTI/Kunitz domain proteins. Finally download the fasta file representing the multiple structure alignment.

**Generation of the Hidden Markow Model for modeling BPTI/Kunitz domain.**

When the alignemnt is returned it is importat to manually check the alignement to look for the conserved residues and the possible errors in the alignement. Ater that you can generate the HMM using *hmmbuild* command from HMMER package.

```
# Generate the HMM
hmmbuild bpti-kunitz.hmm align-bpti-kunitz.fasta
```

**Selection of training and testing set from UniProt.**

This task is performd using the advanced searc of UniProt database using as a key search the Pfam identifier of the BPTI/Kunitz domain. The search has to be restriictued to the protein in SwissProt the are manually annotated. With this search you will obtain ~350 protein containing BPTI/Kuntiz domain that will rapresente the positive set. All the remaining proteins in SwissProt can be used as negatives.

For a fair test of your HMM model you should remove the seqnece in the positive set that share high level of sequence identity with the protein structures collected for generating the model. This task can be performed using *blastpgp* with the option -m 8 checking for sequences with low e-value (column 11) and high sequence identity (column 3).

```
# Run trasforming the sequences of the selected structures in
    ↪ database
formatdb -i selected-bpti-kunits.fasta

# Run blastpgp to matche the positive set agains the selected bpti-
    kunitz.
blastpgp -i positives.fasta -d selected-bpti-kunits.fasta -m 8 -o
    ↪ positives.bl8
```

Using the output of blastpgp you can select the proteins that should be removed from the set of positives.*italicized text*

**Method optimization and assessment.**

When the final benchmark set has been collected a basic testing procedure consist in the implementation of a 2-fold cross-validation test. It consists in splitting positives and negatives in 2 subsets, optimizing the classification threshold on one subset and testing the performance on the other subset.

When the the dataset is splitted in two parts with the same size you can run the *hmmsearch* on the different subsets. It is better to run the command using the following options:

- --max: turns off all the heuristics for cutting of distantly related proteins
- --noali: exclude from the output the alignemnts
- --tblout: returns the output in tabular form
- -Z: It is important for normalizing the e-value output

An example of command for macthing a set of sequences (clean_set.fasta) with the HMM model (bpti-kunitz.hmm) is the follwing:

```
# hmmsearch command matching the model (bpti-kunitz.hmm) with
    ↪ sequence in clean_set.fasta

hmmsearch --noali --max --tblout output.txt -Z 1 bpti-kunitz.hmm
    ↪ clean_set.fasta
```

The output file can be parsed considering the lines without the hash character (#) at the beginning selecting and the column 1 corresponding to the protein identifier and either column 5 or 8 that correspond to the fill-sequence and best-domain e-values respectively.

For the optimization and testing of the performace we need to ckeck the *hmmsearch* output and include those proteins, presumably negatives, for which no match is returned. For the classification purposes you should assign to that proteins and e-value greater or equal than the e-value theshold used for running *hmmsearch*.

When all the results for the subset are collected in files containg protein ID, e-value anc protein class (0/1), we can use the performace.py script previouly developed to calculate the performance of our classification method at different e-value threshold.

# 7 Secondary Structure Propensity

**Exercise 7: Write a python script for calculating the secondary structure propensity od the amino acids.**

For calaculating the amino acid secondary structure propensity we need to select a representative set of protein structures from the PDB.

We can use the advanced search for selecting high quality PDB structures containing only proteins with resolution below 2 .

To remove the redundacy, the clusters of proteins in the PDB are made avalilable at this page.

The information about the secondary structure of the potein can be downloaded from this link. This file contains for each PDB entry its sequence and secondary structure.

After collecting a set of PDB and chains ids we can use a script for parsing fasta files to extract the informtion about the sequence and secondary structure of the proteins in our dataset.

The secondary structure propensity score (PS(aa,ss)) is calculatend comparing the join probability of having a residue aa with secondary structure ss (P(aa,ss)) divided by the probability of independent events (P(aa)P(ss)).

$$PS(aa,ss) = \frac{P(aa,ss)}{P(aa)P(ss)}$$

The python scripts take in input a file containing a list of PDB chain idententifiers (PDBID:CHAIN) and the ss_dis.txt file.

```
#!/usr/bin/python
import sys

def get_ids(idfile):
  ids=open(idfile).read().rstrip().split('\n')
  return ids

def get_count(d):
  daa={}
  dss={}
  das={}
  for k in d.keys():
    for aa in d[k][0]:
      daa[aa]=daa.get(aa,0)+1
    for ss in d[k][1]:
      dss[ss]=dss.get(ss,0)+1
    n=len(d[k][0])
    for i in range(n):
      aa=d[k][0][i]
      ss=d[k][1][i]
      das[(aa,ss)]=das.get((aa,ss),0)+1
  return daa,dss,das

def get_seqs(ids,dbfile):
  d={}
  with open(dbfile, 'r') as fdb:
    for line in fdb:
      if line[0]=='>':
        #pid=line[1:].split(':')[:2]
        pid=line[1:7]
        ts=line[8:].rstrip()
        continue
      if pid in ids:
        d[pid]=d.get(pid,['',''])
        if ts=='sequence': d[pid][0]=d[pid][0]+line.rstrip()
        if ts=='secstr': d[pid][1]=d[pid][1]+line.rstrip('\n').replace(' ','C')
  return d

if __name__ == '__main__':
  idfile=sys.argv[1]
  dbfile=sys.argv[2]
  ids=get_ids(idfile)
  d=get_seqs(ids,dbfile)
  daa,dss,das=get_count(d)
  na=float(sum(daa.values()))
  ns=float(sum(dss.values()))
  nas=float(sum(das.values()))
  for k in das:
    if k[1]!='H': continue
    p=(das[k]/nas)/((daa[k[0]]/na)*(dss[k[1]]/ns))
    print (k,p)
```

The *get_ids* function extrats from the first file the list of identifiers.

The *get_seqs* function generates a dictionary with the PDB chain id as key and a list of two strings including primary sequence and secondary structure as value.

Finally the *get_count* functions generates 3 dictionaries includeing the counting of the amino acid composition (daa) the secondary stuctures (dss) and the join events (das).

The last *if* of the code is included to print only the propensity scale for the -helix.

You can compare the results returned by the previous script with Chou-Fasman propansity scale calculating the correlation coefficient. The linregress function of the scipy.stats library can be used for calculating the correlation coefficient.

# 8 DSSP Data Analysis

*Emidio Capriotti - University of Bologna (Italy)*

Laboratory of Bioinformatics I - Second Module

**Exercise 8: Write a python script for parsing a dssp file.**

Generate a DSSP file from the PDB structure 1BI7 and write a python script to parse the DSSP file. The DSSP file can be genereted using the dssp program. After downloading the dssp program the dssp can be generated using the following command:

```
dssp pdbfile > dsspfile
```

The dssp of the 1BI7 structure can be also downloaded for the ftp repository using the following link.

The script for parsing thd DSSP file starts to extract the information after detecting the header ('# RESIDUE') exluding the row that contain the ! symbol in column 14. The script calculates the relative solvent accessibility of each residue dividing the accessibility by the maximum accessibility of each residue.

The dictionary containing the maximum solvent accessibility of each residue is stored in the Norm_Acc dictionary.

```python
#/usr/bin/python
import sys

Norm_Acc={"A" :106.0, "B" :160.0, \
      "C" :135.0, "D" :163.0, "E" :194.0, \
        "F" :197.0, "G" : 84.0, "H" :184.0, \
      "I" :169.0, "K" :205.0, "L" :164.0, \
      "M" :188.0, "N" :157.0, "P" :136.0, \
        "Q" :198.0, "R" :248.0, "S" :130.0, \
        "T" :142.0, "V" :142.0, "W" :227.0,  \
        "X" :180.0, "Y" :222.0, "Z" :196.0}

def get_dssp(dsspfile):
  dssp=[]
  f=open(dsspfile)
  c=0
  for line in f:
    if line.find('  #  RESIDUE')>-1:
      c=1
      continue
    if c==0: continue
    if len(line)<115: continue
    rnum=line[5:10].strip()
    ch=line[11]
    aa=line[13]
    if aa=='!': continue
    ss=line[16]
    if ss==' ': ss='C'
    acc=float(line[35:38])
    phi=float(line[103:109])
    psi=float(line[109:115])
    rsa=min(1.0,acc/Norm_Acc[aa])
    dssp.append([rnum,ch,aa,ss,rsa,phi,psi])
  return dssp
```

```python

if __name__ == '__main__':
  dsspfile=sys.argv[1]
  dssp=get_dssp(dsspfile)
  print ("#N",'CH','AA','SS','RSA','PHI','PSI',sep='\t')
  for data in dssp:
    (rnum,ch,aa,ss,rsa,phi,psi)=data
    print (rnum,ch,aa,ss,'%.4f' %rsa,'%.1f' %phi,'%.1f' %psi,sep='\
    ↪ t')
```

We can nor run the script to select the data relatives to the amino acid K.

```
!python parse_dssp.py ../data/1bi7.dssp |awk '{if ($3=="K") print
    ↪ $0}'
```

```
26   A  K  E  0.4049   -109.2  163.0
29   A  K  E  0.0146   -96.4   148.5
34   A  K  T  0.4634   -73.5   -33.6
43   A  K  E  0.1854   -127.0  100.3
93   A  K  E  0.4195   -66.9   126.9
111  A  K  H  0.3220   -72.2   -46.8
123  A  K  H  0.3024   -43.3   -41.9
147  A  K  C  0.0244   -79.5   165.8
160  A  K  E  0.0780   -142.4  130.6
216  A  K  S  0.4537   -165.2  128.0
230  A  K  H  0.2537   -35.6   -50.1
257  A  K  C  0.5171   -20.1   122.4
264  A  K  G  0.6244   -61.2   -28.0
274  A  K  H  0.2488   -49.3   -36.1
279  A  K  H  0.4195   -63.8   -25.5
287  A  K  T  0.6634   -79.6   -33.4
```