Sarment Tutorial

Laurent Guéguen

February 2, 2009

The aim of the next sections is to give examples of usage of Sarment. For good comprehension of this tutorial, it is necessary to look into the manual.

The used files are in directory Data, located in this same directory. In the examples, all of the data is in the current directory.

1 Counting on a sequence

In this section, we compute proportions of words on a sequence, from file in specific format lambda.seq, generate randomly a new sequence given these proportions, and compute the likelihood of models on both sequences.

1. From a sequence, we count the 3-length words.

```
import sequence
s=sequence.Sequence(fic="lambda.seq")
import compte
c=compte.Compte()
c.add_seq(s,3)
print c
print c.pref(2)
float(c['ag'])/c['a']
```

2. Afterwards, we compute the proportions of letters, given the letter before. We call them 1|1-proportions.

```
p=compte.Proportion()
p.read_Compte(c,lprior=1,lpost=1)
print p
```

Compare the value for "word" `A|G` with the rate computed before.

3. The end symbol `^` is not relevant for sequence generation, so we remove it.

```
p.read_Compte(c.rstrip(),lprior=1,lpost=1)
print p
```

4. We generate a 10000 letters sequence, given these 1|1-proportions.

```
s2=sequence.Sequence()
s2.read_prop(p,long=10000)
print s2[:10]
```

5. Let's check the 1|1-proportions in `s2`

```
c2=compte.Compte()
c2.add_seq(s2,2)
p2=compte.Proportion()
p2.read_Compte(c2.rstrip(),lprior=1,lpost=1)
print p2
```

6. We translate proportion `p` into a `Lexique`.

```
import descripteur
d=descripteur.Descripteur(1,prop=p)
import lexique
lx=lexique.Lexique()
lx[1]=d
print lx
```

   This rather intricate syntax of `Lexique` is explained in the descriptors part, and the use of it is detailed in this tutorial part.

7. We compute the mean log-likelihood of the markovian model constructed by `p`, on both sequences.

```
print lx.prediction(s)/len(s)
print lx.prediction(s2)/len(s2)
```

   Both are very close.

8. Be careful about the beginning of the sequence. In the next example, a -100000 penalty is due to the lacking proportion for the beginning of the sequence, and the right command to avoid this problem.

```
q=compte.Proportion()
q.read_Compte(c.strip(),lprior=1,lpost=1)
print q          # character ^ has been removed
d.read_prop(q)
lx[1]=d
print lx.prediction(s)
print lx.prediction(s,deb=1)
print lx.prediction(s,deb=1)/(len(s)-1)
```

9. And with longer priors:

```
p3=compte.Proportion()
p3.read_Compte(c.rstrip(),lprior=2,lpost=1)
print p3
d.read_prop(p3)
lx2=lexique.Lexique()
lx2[1]=d
print lx2.prediction(s)/len(s)
print lx2.prediction(s2)/len(s2)
print lx2.prediction(s,deb=2)/(len(s)-2)
print lx2.prediction(s2,deb=2)/(len(s2)-2)
```

Both log-likelihoods are less close than before.

2 Segmenting a sequence

We work on a well-known sequence, the DNA sequence of $\lambda - phage$, in file lambda.seq, which is in specific format. In this example, we use a previously defined HMM, from file lprop1.

1. We load the sequence and the lexique corresponding to the HMM.

```
import sequence
s=sequence.Sequence(fic="lambda.seq")
len(s)
import lexique
lx=lexique.Lexique(fprop="lprop1")
print lx
```

2. We build and draw the partition of the states of the most probable "path" computed with Viterbi algorithm.

```
import partition
p=partition.Partition()
p.viterbi(s,lx)
len(p)
print p
p.draw_nf("lambda1.ps",num=1)
```

3. We compute the Matrice of the log-probabilities of the descriptors given the sequence and the model, by Forward-Backward algorithm.

```
import matrice
m=matrice.Matrice()
m.fb(s,lx)
print m[:10]
```

4. We build a new partition, in which each segment is made where the most probable descriptor in former Matrice is the same on a continuous set of positions.

```
p2=partition.Partition()
p2.read_Matrice(m)
len(p2)
```

5. And if we want to draw the partition such that the height of each arc is proportional with minus the density of log-probability of the model on this segment.

```
p2.draw_nf("lambda_val.ps",num=1,func=lambda x: -x.val()/len(x))
```

6. And we want to draw only the segments described by descriptor 3.

```
p2.draw_nf("lambda_val3.ps",seg=[3],func=lambda x: -x.val()/len(x))
```

7. We compute the proportion of common descriptions between both partitions.

```
float(p2.pts_comm(p))/len(s)
```

8. We build the 50-partitioning from the predictions of the lexique, and draw it.

```
import parti_simp
ps=parti_simp.Parti_simp()
ps.mpp(s,lx,50)
ps.draw_nf("lambda_ps.ps",num=1)
```

9. We compute the list of the similarities between the partitions of ps and both partitions p1 and p2.

```
l=[]
for i in range(len(ps)):
    l.append([i+1,float(ps[i].pts_comm(p))/len(s),\
                  float(ps[i].pts_comm(p2))/len(s)])

for i in l:
    print i
```

Notice and compare the best scores and their corresponding number of segments with the actual number of segments of `p` and `p2`.

3 Building and using HMM

This part is based on data file seq_hmm.fa and Lproportion file lprop2.

We want to compute partitions on a set of sequences from a HMM, and compute proportions from the resulting partitions.

1. We load the sequences and the HMM.

```
import lsequence
ls=lsequence.Lsequence()
ls.read_nf('seq_hmm.fa')
import lcompte
lpr=lcompte.Lproportion(fic="lprop2")
print lpr
```

We notice that in these proportions there is no transition proportion for the sequences beginning.

2. We compute a Lpartition with Viterbi algorithm, and print the number of segments of each partition

```
import lpartition
lpa=lpartition.Lpartition()
lpa.add_Lseq(ls)
import lexique
lx=lexique.Lexique()
lx.read_Lprop(lpr)
```

```
lpa.viterbi(lx)
for i in lpa:
    print len(i[1]),
```

3. We build a new Lcompte of 2-length words from the segmentations.

```
lc=lcompte.Lcompte()
lc.read_Lpart(lpa,2)
print lc
```

4. We compute a new HMM from the 1|1-proportions on the latter Lcompte.

```
lpr2=lcompte.Lproportion()
lpr2.read_Lcompte(lc.rstrip(),lprior=1,lpost=1)
```

5. We compute the Kullback-Leibler divergence form `lpr` to `lpr2`, using MC simulation on 100 sequences of length 5000.

```
lpr.KL_MC(lpr2,100,5000)
```

6. On the studied sequences, we compute new partitions with this new HMM with FB algorithm.

```
lx2=lexique.Lexique()
lx2.read_Lprop(lpr2)
lpa2=lpartition.Lpartition()
lpa2.add_Lseq(lpa.Lseq())
lpa2.fb(lx2)
```

7. And so on.

4 Usage of Lexique and Parti_simp

We give several examples of Lexique, to show how use its specific syntax. These examples are used on the DNA sequence of $\lambda - phage$, in file lambda.seq, which is known to show strong signals of segmentation [Chu92, Chu89, Gué00].

In connection with Lexique, we give examples of computation and manipulation of partitionnings.

1. We load the sequence.

```
import sequence
s=sequence.Sequence(fic="lambda.seq")
```

2. We want to part this sequence such that a segment is described by a letter, and its value is the number of occurences of this letter.

```
print s.alpha()
import lexique
lx1=lexique.Lexique()
lx1.read_str('0:a 1:c 2:g 3:t')
print lx1
import parti_simp
ps1=parti_simp.Parti_simp()
ps1.mpp(s,lx1,100)
```

3. We draw the even-numbered partitions.

```
ps1.filter(lambda x: len(x)%2==0).draw_nf("lambda1.ps")
```

4. We cluster the segments described by `c` or `g`, and the ones described by `a` or `t`, and sort the new `Parti_simp` following the number of segments. We draw the 20 first partitions of this new `Parti_simp`.

```
ps2=ps1.group([[1,2],[0,3]])
ps2.sort()
ps2[:20].draw_nf("lambda1_cg.ps",seg=[1,2])
```

5. We want to partition on comparing the occurences in `c` and `g` versus the ones in `a` and `t`.

```
lx2=lexique.Lexique(str="0:+(cg) 1:+(at)")
ps2=parti_simp.Parti_simp()
ps2.mpp(s,lx2,20)
ps2[:20].draw_nf("lambda2_cg.ps",num=1)
```

6.

5 Simulations with Bernoulli models

In this section, we perform simulations of sequences build a Bernoulli models, and compute the log-likelihoods on them.

```
import partition
import random
import sequence
import lcompte
```

```
import matrice
import parti_simp
import compte
import lexique
```

1. The function that generates a Bernoulli model:

```
def gen_bern(lalpha):
    lpr=lcompte.Lproportion()
    for i in range(len(lalpha)):
        alpha=lalpha[i]
        p = compte.Proportion()
        s="|A␣%f\n|B␣%f\n"%(alpha,1-alpha)
        p.read_str(s)
        lpr[i]=p

    return lpr
```

2. We define some parameters

```
lalpha=[0.3,0.7]  # parameters of the models
lseq=10000 #length of the sequences

nsegprob=100 # length of the computation
nseg=15   # nb of simulated segments

nbsampl=100  # nb of samples
```

3. and generate the Lexique.

```
ldesc=lpr.num()
lldesc=len(ldesc)
if lldesc<=1:
    print "Trop␣few␣models"
    exit

lx=lexique.Lexique(Lprop=lpr)
```

4. The output file:

```
s=reduce(lambda x,y: x+"_"+y, map(str, lalpha))
fprob=open("bern_%s_n%d.llh"%(s,nseg),"w")
```

```
for i in range(nsegprob):
    fprob.write("N%d\t"%(i+1))
fprob.write("\n")
```

5. And the simulation loop:

```
for i in range(nbsampl):
    p=partition.Partition()    #generation of the partition
    p.build_random(lseq,nseg,ec=50)
    ns=random.choice(ldesc)
    for sg in p:
        sg.g_num([ns])
        ns2=ns
        while ns2==ns:
            ns=random.choice(ldesc)

    s=sequence.Sequence()
    s.generate(lseq)
    for sg in p:
        s.read_prop(lpr[sg.num()[0]],deb=sg.deb(),fin=sg.fin())

    ################
    # Log-likelihoods
    lp=lx.log_likelihood(s,nsegprob)

    ################
    # output
    for x in lp:
        fprob.write("%.12f\t"%(x))
    fprob.write("\n")

    fprob.flush()

fprob.close()
```

6 Predicting the number of classes

In this section, we build a random sequence, on a given number of classes, from a set of markovian models, and we use the probability algorithm to predict the given number.

In this example, the model is in file lprop1.

1. We build a Partition, on a virtual sequence of length 10000, in 23 segments, such that each segment is at least 50 positions long.

```
nbcl=23 #for example
import lcompte
lp=lcompte.Lproportion(fic="lprop1")
import partition
p=partition.Partition()
p.build_random(10000,nbcl,ec=50)
```

2. We attribute predictor numbers to the Segments of the Partition.

```
lnum=lp.num()
import random
numr=random.choice(lnum)
for s in p:
    s.g_num([numr])
    x=random.choice(lnum)
    while x==numr:
        x=random.choice(lnum)
    numr=x
```

3. Then we build a Sequence from the Partition and the Lproportion.

```
import sequence
s=sequence.Sequence()
s.read_Part(p,lp)
```

4. We build a Matrice from the predictions of the models.

```
import lexique
import matrice
lx=lexique.Lexique(Lprop=lp)
```

5. We compute the segmentation log-likelihood, up to 100 classes.

```
lprob=lx.log_likelihood(s,100)
```

6. We add the exponential a priori:

```
import math
lapost=[]
theta=0.514
```

```
for i in range(len(lprob)):
    lapost.append(lprob[i]+(i+1)*math.log(theta))
print
```

7. We select the maximum of the ratio.

```
nbc=lapost.index(max(lapost))+1
print nbc
```

7 Looking for CpG islands

In mammal genomes, methylation of cytosine in dinucleotides CG (called CpG) entails low frequency of such dinucleotides, excepted in small segments, called *CpG islands* (see [PDM01]). A usual way to detect CpG islands is to compute the ratio $\frac{\text{number of observed CpG}}{\text{number of expected CpG}}$ on sliding windows.

To goal of this part is to segment a mouse sequence to reveal CpG islands. The modelling used has two states: model of CpG island vs model of methylated sequence. Both models are learned from the proportions of words in sets of sequences, respectively in files mus_cpg.fa and mus_tem.fa.

First, we use the MPP algorithm and compute the segmentation probability to build a maximum prediction partition.

After, we proceed an HMM analysis. For this, we introduce the information that in mouse genome, CpG islands are in average 1000 bases long, and are spaced by in average 125,000 bases.

At the end, we draw the partitions computed by these methods with some computed by CpG specialised methods.

1. We count the 1|1-proportions on the Lsequence built from both sets of sequences.

```
import lsequence
ls_cpg=lsequence.Lsequence(fic="mus_cpg.fa")
ls_tem=lsequence.Lsequence(fic="mus_tem.fa")
import compte
c_cpg=compte.Compte()
for s in ls_cpg:
    c_cpg.add_seq(s,2)

c_tem=compte.Compte()
for s in ls_tem:
    c_tem.add_seq(s,2)

p_cpg=c_cpg.strip().prop(lprior=1,lpost=1)
```

```
print p_cpg
p_tem=c_tem.strip().prop(lprior=1,lpost=1)
print p_tem
```

2. We have to remove the `N` letters:

```
p_cpg=c_cpg.restrict_to('ACGT').strip().prop(lprior=1,lpost=1)
print p_cpg
p_tem=c_tem.restrict_to('ACGT').strip().prop(lprior=1,lpost=1)
print p_tem
```

3. Then we build a Lproportion for our modelling. We introduce the inter-state probabilities for the HMM:

```
import lcompte
lp=lcompte.Lproportion()
lp[0]=p_cpg          # state 0 for cpg islands
lp[1]=p_tem          # state 1 for sequences between of cpg islands
lp.g_inter(0,0,1-1/1000.0)
lp.g_inter(0,1,1/1000.0)
lp.g_inter(1,0,1/125000.0)
lp.g_inter(1,1,1-1/125000.0)
print lp
```

4. To build partitions of the studied sequence, we have to use a Lexique.

```
import sequence
s_mus=sequence.Sequence(fic="mus2.fa")
import lexique
lx_hmm=lexique.Lexique()
lx_hmm.read_Lprop(lp)
```

5. We compute the segmentation likelihood of the sequence from the models used to build the sequence, up to 200 classes.

```
lprob=lx_hmm.log_likelihood(s_mus,100)
```

6. We want to compute the variance of the distribution of the probabilities of the Sequence.

   - We build a Matrice, corresponding to the predictions of the Lexique `lx_hmm` on the Sequence.

```
import matrice
m=matrice.Matrice()
m.prediction(s_mus,lx_hmm)
```

- We build the Lexique for the 2nd moment, and compute it. After we compute the variance of the probabilities.

```
import math
lx2=lexique.Lexique(str="0:#0(2)_1:#1(2)")
lprob2=lx2.log_likelihood(m,100)

lvar=[]
for i in range(len(lprob)):
    lvar.append(lprob2[i]+math.log(1+math.exp(2*lprob[i]-lprob2[i])))
```

7. We compute the MPP corresponding to the Lexique.; and we draw it, with the height of each arc proportional with the G+C density.

```
import parti_simp
ps=parti_simp.Parti_simp()
ps.mpp(s_mus, lx_hmm, 50)
```

8. We want to draw the partitioning such that the height of each arc is proportional to the density in C+G. So, we use a Lexique with the descriptor C+G.

```
lx_cg=lexique.Lexique(str="+(CG)")
ps.draw_nf("mus_ps_cg.ps",\
```

9. We compute the *a posteriori* log-probabilities, with an *a priori* $P(\mathbb{P}_k|D) \propto 0.546^k$, and select the partition of the maximal estimator:

```
lap=[]
for i in range(len(lprob)):
    lap.append(lprob[i]+(i+1)*math.log(0.546))

nbc=lap.index(max(lap))+1
p_nbc=ps[nbc-1]
```

10. The next partition is computed via Viterbi algorithm.

```
import partition
p_vit=partition.Partition()
p_vit.viterbi(s_mus,lx_hmm)
print p_vit
```

11. The following one is built via forward-backward algorithm, by the most probable state on each position. Then, before, the Matrice of the log-probabilities of the states must be computed.

```
import matrice
m_fb=matrice.Matrice()
m_fb.fb(s_mus,lx_hmm)
p_fb=partition.Partition()
p_fb.read_Matrice(m_fb)
print p_fb
```

12. And we draw the partitions:

```
p_nbc.draw_nf("mus_nbc_cg.ps",func=lambda s: \
                lx_cg.prediction(s_mus,deb=s.deb(),fin=s.fin())/len(s))
p_vit.draw_nf("mus_vit_cg.ps",func=lambda s: \
                lx_cg.prediction(s_mus,deb=s.deb(),fin=s.fin())/len(s))
p_fb.draw_nf("mus_fb_cg.ps",func=lambda s: \
               lx_cg.prediction(s_mus,deb=s.deb(),fin=s.fin())/len(s))
```

13. We want to draw these partitions such that the height of each arc is proportional to the value:

$$\frac{\text{number of observed CpG}}{\text{number of expected CpG}} = \frac{\text{number of CpG.length of the segment}}{\text{number of C.number of G}}$$

We use a Lexique with 3 descriptors: C G and word CG, and define the function on a lexique, a sequence, and a segment, that returns the ratio.

```
lx_oe=lexique.Lexique(str="0:C␣1:G␣2:'CG'")
def height(l,sq, sg):
    dl=l.ls_evalue(sq,deb=sg.deb(),fin=sg.fin())
    if dl[1]*dl[0]==0:
        return 0
    else:
        return float(dl[2]*len(sg))/(dl[1]*dl[0])
```

And we draw the partitions:

```
p_nbc.draw_nf("mus_nbc_oe.ps",func=lambda s: height(lx_oe,s_mus,s))
p_vit.draw_nf("mus_vit_oe.ps",func=lambda s: height(lx_oe,s_mus,s))
p_fb.draw_nf("mus_fb_oe.ps",func=lambda s: height(lx_oe,s_mus,s))
```

14. We want to compare those methods with some specifically made for CpG islands detection: `CpGproD` [PM01], `CpGplot` [LGLP92] and `CpG Island Searcher` [TJ02]. As the outputs of these programs have different syntaxes, they are converted in several files. Those files are, respectively, cpgprod.part, cpgplot.part, and cpgis.part.

    They are all put un a `Parti_simp`, with the ones computed with HMM algorithms:

```
pscmp=parti_simp.Parti_simp()
pscmp.append(partition.Partition(fic="cpgprod.part"))
pscmp[-1].s_name("cpgprod")
pscmp.append(partition.Partition(fic="cpgplot.part"))
pscmp[-1].s_name("cpgplot")
pscmp.append(partition.Partition(fic="cpgis.part"))
pscmp[-1].s_name("cpgis")
pscmp.append(p_fb)
pscmp[-1].s_name("fb")
pscmp.append(p_vit)
pscmp[-1].s_name("vit")
pscmp.append(p_nbc)
pscmp[-1].s_name("mpp")
pscmp.draw_nf("tous_cpg.ps",seg=[0])
```

References

[Chu89] G.A. Churchill. Stochastic models for heterogenous DNA sequences. *Bulletin of Mathematical Biology*, 51(1):79–94, 1989.

[Chu92] G.A. Churchill. Hidden Markov chains and the analysis of genome structure. *Computers Chem.*, 16(2):107–115, 1992.

[Gué00] L. Guéguen. *Partitionnement maximalement prédictif sous contrainte d'ordre total. Applications aux séquences génétiques.* Thèse, Université Pierre et Marie Curie - Paris VI, janvier 2000.

[LGLP92] F. Larsen, G. Gundersen, R. Lopez, and H. Prydz. CpG islands as gene markers in the human genome. *Genomics*, 13(4):1095–107, 1992.

[PDM01] L. Ponger, L. Duret, and D. Mouchiroud. Determinants of cpg islands: expression in early embryo and isochore structure. *Genome Res.*, 11(11):1854–1860, 2001.

[PM01] L. Ponger and D. Mouchiroud. CpGProD: identifying CpG islands associated with transcription start sites in large genomic mammalian sequences. *Bioinformatics*, 18:631–633, 2001.

[TJ02] D. Takai and P.A. Jones. Comprehensive analysis of cpg islands in human chromosomes 21 and 22. *PNAS*, 99(6):3740–3745, 2002.