RNA-seq 표준 분석 프로토콜 (SOP)

2016.10.14

한양대학교 생명과학과
남진우 교수

해당 RNA-seq Standard Operation Protocol (SOP)은 기본적인 RNA-seq 분석 프로토콜을 목적으로 하여 설계되었음. 여기서 다루고 있는 RNA-seq 시퀀싱 데이터는 Illumina 플랫폼에 의해 생산된 시퀀싱 데이터를 기초로 하고 있으며, 개별 연구자의 분석 목적에 따라 프로그램 별 세부 파라미터의 수정이 필요할 뿐만 아니라 시퀀싱 플랫폼 (플랫폼 별 error-rate)과 시퀀싱 디자인 (샘플 수, depth)에 따라 많은 변동사항이 있기 때문에 해당 SOP 에서 설명한 프로토콜은 모든 연구를 대변하는 방법론이 될 수 없음. 또한 현재 구축된 프로토콜은 최적화하는 단계를 통해 추후 변경될 가능성이 있음 (RNA-seq SOP 는 2016.10.14 현재 기준으로 작성되었음.).

목 차

1. 배경

RNA 의 동정과 발현은 RNA 의 중간자 및 조절인자로서의 역할이 알려진 후 분자생물학에서 가장 중요한 활동 중 하나가 되었음. 현재 가장 널리 쓰이는 RNA sequencing 은 동정과 정량의 두 가지 목적을 동시에 달성할 수 있기 때문에 이전의 microarray 를 빠르게 대체했음. 또한 RNA-seq 은 방대한 활용도 때문에 다양한 변형을 거쳐 특정 조건의 RNA 조각들에 대한 연구를 가능케 했음. 이러한 변형들로 연구할 수 있는 주제는 alternative splicing, gene fusion, transcript boundary variation, structure variation, differentially expressed gene (DEG) 등 생명과학 분야 전체에 걸쳐 있음. 특히 DEG 분석은 RNA-seq 을 이용한 분석에서 가장 기초가 되는 작업으로, 각기 다른 조건의 샘플에서 발현량 차이를 보이는 유전자들을 골라내는 것을 말함. 이를 위해서는 먼저 준비한 RNA-seq 데이터를 매핑하여 발현량을 예측하고, 유전자마다 샘플간의 발현량이 유의하게 다른지 알아봄. 분석이 끝나면 얻은 유전자 집합을 다양한 추후 분석에 사용할 수 있음 (그림 1).

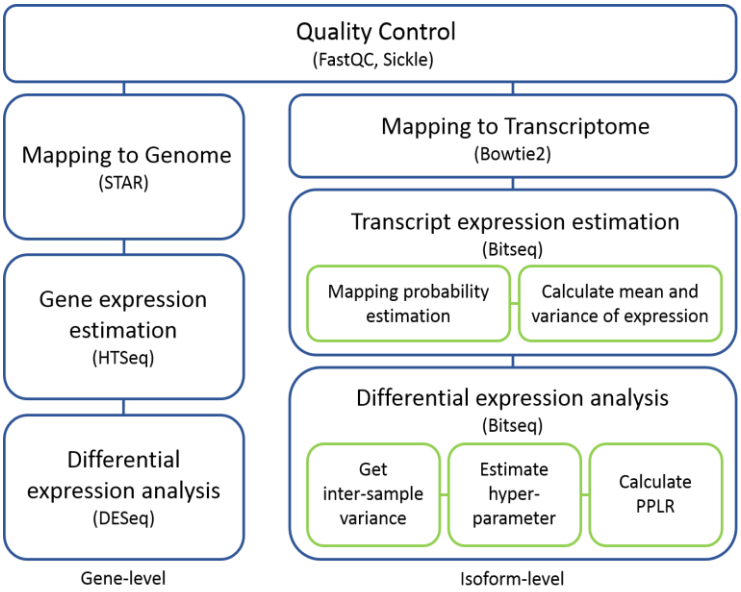

그림 1 RNA-seq 분석의 모식도

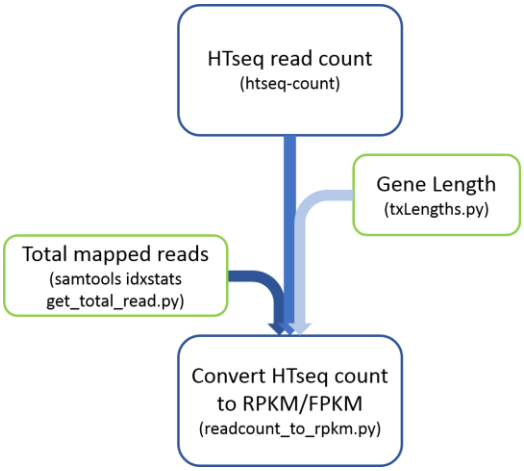

그림 2 HTSeq 세부 단계의 모식도

2. 준비

1) 원시데이터 준비

- 내용: 분석할 RNA-seq 데이터 파일을 준비함. mRNA-seq 이라도 sequencing 을 진행한 방법에 따라 여러 종류로 분류될 수 있음. 분류 기준은 주로 해당 sequencing 이 single-end 또는 paired-end 로 되었는지, 그리고 RNA 의 strand 정보가 유지되었는지 손실되었는지 등으로 나눔. 종류 외에도 read 의 길이나 얼마나 많이 sequencing 이 되었는지 (sequencing depth) 등의 정보를 확인해야 함.
- Read 의 길이: 50~150nt
- Sequencing depth: > 3G
- Library type: Strand-specific, single-end/paired-end
- Sequencing platform: Illumina

2) 프로그램 준비

- FastQC[1] 설치
- Sickle[2] 설치
- STAR[3] 설치
- RSeQC[4] 설치
- Python[5] 설치
- HTSeq[6] 설치
- Samtools[7] 설치
- R[8] 설치
- DESeq[9] 설치
- Bowtie2[11] 설치
- BitSeq[12] 설치

3. 원시 데이터 품질 보정

1) FastQC 로 원시 데이터의 품질 확인

- 내용: FastQC 를 이용하여 원시 데이터의 품질, 오염 정도, 길이 분포, 각 염기별 포함 정도 등을 확인함.
- 명령어의 예 (FastQC 프로그램)

```
fastqc -o [output directory] -f fastq data1_1.fastq data1_2.fastq
```

- 옵션의 설명

  -o: 결과물을 생성하고 저장할 디렉토리

  -f: 입력 데이터파일의 종류 (bam, sam 과 fastq 지원)

- 결과물

  - fastqc_data1_1.txt : RNA-seq read 의 품질 정보가 쓰여진 파일.

```
##FastQC       0.10.1
>>Basic Statistics     pass
#Measure       Value
Filename       GSM546921_filtered_sequence.txt
File type      Conventional base calls
```

  - fastqc_report.html: fastqc_data1_1.txt 의 내용을 웹 페이지로 볼 수 있는 파일.
  - summary.txt: fastqc_data1_1.txt 의 품질검사 단계에 따른 최종 결과 (PASS/WARN/FAIL)가 적힌 파일

```
PASS    Basic Statistics       GSM546921_filtered_sequence.txt
PASS    Per base sequence quality      GSM546921_filtered_sequence.txt
PASS    Per sequence quality scores    GSM546921_filtered_sequence.txt
WARN    Per base sequence content      GSM546921_filtered_sequence.txt
```

2) Sickle 로 원시 데이터의 품질 관리

- 내용: FastQC 로 원시 데이터의 품질을 확인하고, Sickle 로 원시 데이터의 품질을 관리함.

- 기준
  - 윈도우 내 base quality score 의 평균: 20 이상
  - 최소 read 길이: 20 이상

- 명령어의 예 (Sickle 프로그램)

```
sickle pe -f data1_1.fastq -r data1_2.fastq -t sanger -o
data1_1.trimmed.fastq -p data1_2.trimmed.fastq -s
data1_single.trimmed.fastq -q 20 -l 20
```

- 옵션의 설명

  se/pe: 원시 데이터의 타입이 single-end 이면 se, paired-end 이면 pe 로 설정
  -f: 입력 데이터의 첫번째 read 파일
  -r: 입력 데이터의 두번째 read 파일 (원시 데이터의 타입이 paired-end 인 경우에만 해당됨)
  -t: base quality 인코딩 타입에 따라 설정 (solexa (CASAVA < 1.3), illumina (CASAVA 1.3 to 1.7), sanger (CASAVA >=1.8))
  -o: 첫번째 read 파일의 결과물을 생성하고 저장할 파일 이름
  -p: 두번째 read 파일의 결과물을 생성하고 저장할 파일 이름 (원시 데이터의 타입이 paired-end 인 경우에만 해당됨)
  -s: single-end 로 변환되는 결과물을 생성하고 저장할 파일 이름 (원시 데이터의 타입이 paired-end 인 경우에만 해당됨)
  -q: window 내 base quality score 의 평균 기준
  -l: 최소 read 길이 기준

- 결과물
  - data1_1.trimmed.fastq: 품질 관리된 data1_1.fastq 파일
  - data1_2.trimmed.fastq: 품질 관리된 data1_2.fastq 파일
  - data1_single.trimmed.fastq: 품질 관리 후 pair 중 한 쪽만 남은 read 들의 파일

4. 참조 유전체에 RNA-seq 데이터 alignment

- 내용: 품질 관리된 시퀀싱 데이터를 STAR 를 통해 참조 유전체에 맵핑.

- 파일 준비
  - FASTQ 포맷의 품질 관리된 시퀀싱 데이터 파일 (예: data1_1.trimmed.fastq)
  - FASTA 포맷의 참조 유전체 데이터 파일 (예: Homo_sapiens.GRCh37.chromosomes.chr.fa)

1) 참조 유전체 데이터 파일 다운로드

● 내용: 시퀀싱 데이터를 맵핑할 참조 유전체 데이터 파일의 다운로드.

● 방법

- 참조 유전체[13] 데이터 다운로드 페이지에 접속.
- 해당 생물 종의 참조 유전체 버전에 해당하는 데이터를 선택.
- FASTA 포맷의 "Homo_sapiens.GRCh37.chromosomes.chr.fa" 파일을 원하는 위치에 다운로드 (인간 참조 유전체의 GRCh37 버전 파일)

2) 참조 유전체 데이터 파일의 인덱스 생성

● 내용: 맵핑에 사용할 참조 유전체 데이터 파일의 인덱스 생성.

● 명령어의 예 (STAR 프로그램)

```
STAR --runMode genomeGenerate --genomeDir [output directory]
--genomeFastaFiles Homo_sapiens.GRCh37.chromosomes.chr.fa
```

옵션의 설명

--runMode: STAR 프로그램의 실행모드 (인덱스 생성은 "genomeGenerate"로 설정함)

--genomeDir: 참조 유전체 데이터 파일의 인덱스를 생성하고 저장할 디렉토리

--genomeFastaFiles: 참조 유전체 서열 데이터 파일 (FASTA 포맷)

● 결과물

- 참조 유전체 서열 데이터의 인덱스 파일들

3) 참조 유전체에 시퀀싱 데이터 맵핑

● 내용

- STAR 로 품질 관리된 시퀀싱 데이터를 참조 유전체 데이터에 맵핑.
- STAR 는 짧은 길이의 리드로 이루어진 시퀀싱 데이터를 정확하고 빠르게 맵핑함.
- 인덱스 파일은 위 과정 중에서 STAR 의 genomeGenerate 로 만든 파일을 얘기하며, 맵핑 명령어 입력 시 인덱스 저장 디렉토리를 입력함.
- 시퀀싱 데이터가 paired-end 인 경우, 공백으로 첫번째 read 입력 파일과 두번째 read 입력 입력 파일을 구분함.
- STAR 의 결과물로 BAM 포맷의 맵핑된 데이터를 얻음.

● 명령어의 예 (STAR 프로그램)

```
STAR --runMode alignReads --runThreadN 2 --outFilterMultimapNmax 10 --
alignIntronMin 61 --alignIntronMax 265006 --genomeDir /인덱스_파일_디렉토리/
--readFilesIn data1_1.trimmed.fastq data1_2.trimmed.fastq --outSAMtype
BAM SortedByCoordinate --outFileNamePrefix [output filename]
```

● 옵션의 설명

--runMode: STAR 프로그램의 실행모드 (맵핑은 "alignReads"로 설정함)

--runThreadN: 맵핑에 사용할 thread 개수

--outFilterMultimapNmax: 결과물 (BAM 포맷의 맵핑된 데이터)에 기록 가능한 각 read 별 최대 맵핑 허용치 (허용치를 초과할 경우 맵핑 되지 않는 것으로 판단함)

--alignIntronMin: 맵핑 허용 최소 인트론 길이 (본 내용에서는 알려진 인간 유전자의 인트론 길이 분포로부터 0.01% 구간을 설정함)

--alignIntronMax: 맵핑 허용 최대 인트론 길이 (본 내용에서는 알려진 인간 유전자의 인트론 길이 분포로부터 99.9% 구간을 설정함)

--genomeDir: 맵핑에 사용할 참조 유전체 인덱스가 저장된 디렉토리

--readFilesIn: 품질 관리된 입력 데이터 파일 (입력 데이터의 타입이 paired-end 인 경우는 공백으로 첫번째 read 와 두번째 read 파일을 구분하여 입력)

--outSAMtype: 출력 데이터 파일의 종류 (SAM: SAM 파일, BAM Unsorted: 포지션으로 정돈되지 않은 BAM 파일, BAM SortedByCoordinate: 포지션으로 정돈된 BAM 파일)

--outFileNamePrefix: 맵핑된 결과물을 생성하고 저장할 파일 이름의 접두사를 지정

● 결과물

- Aligned.sortedByCoord.out.bam: 맵핑된 결과물이 저장된 BAM 파일
- Log.final.out: 맵핑된 결과물의 다양한 통계결과를 확인할 수 있는 파일

```
        Number of input reads |      24330327
    Average input read length |      202
                  UNIQUE READS:
Uniquely mapped reads number |      22593408
```

- Log.out: 맵핑 진행 옵션의 로그가 저장된 파일
- Log.progress.out: 맵핑 진행 상황의 로그가 저장된 파일
- SJ.out.tab: 맵핑된 Splice-Junction (SJ) 정보가 저장된 파일 (탭으로 분리된 파일)

```
chr1   14830  14969  2      2      1      3      5      38
chr1   15039  15795  2      2      1      2      5      32
chr1   16766  16857  2      2      1      0      1      41
```

4) 맵핑의 품질 검사

● 내용

- 맵핑의 결과로 생성된 BAM 또는 SAM 파일의 정보를 통해 맵핑 결과의 여러 기본 통계값을 구해봄.
- RSeQC 는 시중에 무료로 배포되어 있는 python 프로그램으로, RNA-seq 과 관련된 여러 기본 기능들을 지원함. 이 중 bam_stat.py 기능은 BAM 또는 SAM 파일에서 지정한 mapping quality

보다 더 품질이 낮은 aligment 가 얼마나 있는지 보여줌.

- Mapping quality 는 "해당 read 가 align 된 region 에서 기원하지 않았을 확률"에 $-10\log_{10}$ 을 붙여서 나타내는데, 만약 mapping quality 가 30 이라면 read 가 실제로 해당 region 에서 만들어지지 않았을 확률은 0.001 이 됨.

- 명령어의 예 (bam_stat.py)

```
bam_stat.py -i data1.bam
```

- 옵션의 설명

  -i: 맵핑으로 생성된 SAM 또는 BAM 파일

  -q: mapping quality (default 30)

- 결과물

  - 모든 데이터 값은 read 개수로 표현되며, 모든 통계치의 상세한 설명은 RSeQC 페이지에 나와 있음
  - Total records: 전체 mapping 된 read 개수
  - QC failed: QC 에서 품질이 낮아 걸러진 read 개수
  - Non primary hits: best hit 이 아닌 다른 alignment 의 개수
  - Unmapped reads: mapping 이 되지 않은 read 개수
  - mapq < mapq_cut (non-unique): mapping quality 가 기준 미만으로, unique 하다고 볼 수 없는 read 개수
  - mapq >= mapq_cut (unique): mapping quality 가 기준 이상으로, unique 하다고 볼 수 있는 read 개수

```
Total records:                          16354
QC failed:                              0
Optical/PCR duplicate:                    0
Non primary hits                        11384
Unmapped reads:                         272
mapq < mapq_cut (non-unique):             3186
mapq >= mapq_cut (unique):                1512
```

5. 유전자 발현량 측정

- 내용: HTSeq 과 스크립트를 이용해 맵핑된 데이터를 각 유전자의 발현량 (RPKM/FPKM)으로 변환함.

1) 참조 유전체에 시퀀싱 데이터 맵핑

- 내용
  - HTSeq 을 이용해 맵핑된 데이터를 각 유전자에 맵핑된 read 수로 변환함.
  - HTSeq 은 유전자가 가지고 있는 모든 exon 에 대해, 사용자가 원하는 조건에 부합하는 RNA-seq read 의 개수를 세어 주는 프로그램임.
  - HTSeq 은 아주 빠르고 결과로 유전자에 할당된 RNA-seq read 의 개수를 출력하기 때문에 활용 범위가 넓음.

- HTSeq 은 따라서 이소체의 발현량을 측정하기에는 적합하지 않음.

- 명령어의 예 (HTSeq 프로그램)

```
htseq-count -s reverse -m intersection-nonempty -f bam data1.bam
gencode.v19.anntoation.gtf[12)] > data1_htseq_count.txt
```

- 옵션의 설명

  -f: 맵핑된 입력 데이터의 포맷 (sam/bam)

  -r: 시퀀싱 데이터 타입이 paired-end 인 경우, 입력파일이 어떤 방식으로 정렬되어 있는지 입력 (pos: 리드의 포지션, name: 리드의 이름)

  -s: 시퀀싱 데이터가 strand-specific 프로토콜을 사용했는지 입력 (yes/no/reverse)

  -a: 맵핑 데이터로부터 read 수로 변환할 때 고려되는 최소한의 맵핑 quality

  -m: 여러 구간에 걸쳐 맵핑된 read 를 고려하는 방법 (union/intersection-strict/ intersection-noempty)

- 결과물

  - data1_htseq_count.txt: 각 유전자마다 맵핑된 read 의 개수가 계산된 파일

```
ENSG00000008735.10      5
ENSG00000015475.14      448
ENSG00000025708.8       460
ENSG00000025770.14      341
ENSG00000040608.9       2
```

2) HTSeq 에서 쓰인 유전자의 길이 계산 및 total mapped read 구하기

- 내용

  - HTSeq 의 결과로 얻은 RNA-seq 의 read count 를 유전자의 발현량의 한 단위로 쓰이는 RPKM/FPKM 단위로 변환하기 위해서는 유전자의 길이를 먼저 계산해야 함.
  - HTSeq 은 유전자의 모든 exon 을 사용하여 RNA-seq 의 read count 를 계산함.
  - 따라서 유전자가 가지는 모든 exon 의 길이를 합친 데이터를 만들어야 함.
  - RPKM/FPKM 을 구하기 위해서는 mapping 된 모든 read 의 개수도 알아야 함.
  - total mapped read 개수를 구하기 위해 samtools idxstats 를 사용함.
  - idxstats 는 각 염색체별로 염색체의 길이, mapping 된 read 의 개수, mapping 되지 않은 read 의 개수를 보여줌.

- 명령어의 예 (in-house Python script)

```
python txLength.py gencode.v19.annotation.gtf /home/name_of_user/
infile = sys.argv[1]     #HTSeq에 썼던 gtf 파일
outdir = sys.argv[2]     #결과물을 저장할 디렉토리
exon_dict = {}
merged_exon_dict = {}    #겹치는 exon들을 통합하여 저장할 변수
inopen = open(infile)
for line in inopen:
   linesp = line.split('\t')
      if linesp[2] == 'exon' :     #gtf 파일 중 exon에 해당하는 줄
         start = int(linesp[3])    #exon의 start 위치
         end = int(linesp[4])      #exon의 end 위치
         attribute = linesp[8].strip('\n').split('; ')
         gene_id = attribute[0].split('"')[-2]      #gene ID를 저장
         if not exon_dict.has_key(gene_id) : exon_dict[gene_id] = []
         exon_dict[gene_id].append((start, end))    # gene ID 마다 exon의 위치 정보를 저장
inopen.close()

for gene in exon_dict.keys():
   exons = exon_dict[gene]        #gene ID에 해당하는 모든 exon들
   exon_num = len(exons)
   merged_exon_dict[gene] = []
      if exon_num == 1 : merged_exon_dict[gene] = exons   #exon이 하나밖에 없으면 그대로 넣음
      for i in range(exon_num-1):
         this_exon = exons[i]      #순서대로 exon을 지정
         overlapping = []
         for j in range(i+1, exon_num):
            other_exon = exons[j]  #다른 exon을 돌아가며 지정
            overlap_check = min(this_exon[1], other_exon[1]) - max(this_exon[0],
other_exon[0])
            if overlap_check >= 0 : overlapping.append(other_exon) #두 exon이 겹치면 다른 변수에
모아 둠
            else: pass
         if len(overlapping) > 0: #겹치는 exon이 존재
            all_start = map(lambda x: x[0], overlapping)
            all_end = map(lambda x: x[1], overlapping)
            new_exon = (min(all_start), max(all_end))
            merged_exon_dict[gene].append(new_exon) #모든 exon들을 합침
         else: merged_exon_dict[gene].append(this_exon)
outfile = outdir + 'geneLength.txt'
outopen = open(outfile, 'w')
for genes in merged_exon_dict.keys():
       merged_exons = merged_exon_dict[genes]
       exon_lengths = map(lambda x: abs(x[1]-x[0]), merged_exons)
       gene_length = sum(exon_lengths)    #exon들의 길이를 합침
       outopen.write(genes + '\t' + str(gene_length) + '\n')       #결과 파일에 출력
outopen.close()
```

- 결과물

  - geneLength.txt: 각 유전자마다 HTSeq에 사용된 유전자의 길이가 계산된 파일

```
ENSG00000099942.8       6804
ENSG00000100243.16      38997
ENSG00000215012.4       22778
ENSG00000233010.1       601
```

3) Mapping 된 bam 파일의 total mapped read 구하기

- 내용:
  - HTSeq 의 결과로 얻은 RNA-seq 의 read count 를 유전자의 발현량의 한 단위로 쓰이는 RPKM/FPKM 단위로 변환하기 위해서 mapping 된 모든 read 의 개수도 알아야 함.
  - Total mapped read 개수를 구하기 위해 samtools idxstats 를 사용함.
  - idxstats 는 각 염색체별로 염색체의 길이, mapping 된 read 의 개수, mapping 되지 않은 read 의 개수를 보여줌.
  - idxstats 는 read 의 indexing 을 통해 전체적인 수치를 보여주는 도구임.
  - idxstats 를 사용하기 위해서는 먼저 bam 파일을 정렬하고 indexing 을 해야 함.

- 명령어의 예 (samtools)

```
samtools sort data1.bam data1_sort.bam    #bam 파일을 유전체상 위치대로 정렬
samtools index data1_sort.bam     #idxstats 를 이용하기 위해 indexing 을 함
samtools idxstats data1_sort.bam > idx.tx
```

- 결과물
  - idx.txt: 순서대로 염색체의 길이, mapping 된 read 의 개수, mapping 되지 않은 read 의 개수임. mapping 된 read 의 개수를 모두 더하면 total mapped read 를 구할 수 있음.

```
chr1   249250621     538885 0
chr10  135534747     130018 0
chr11  135006516     312722 0
chr12  133851895     291136 0
```

- 명령어의 예 (in-house Python script)

```
python get_total_read.py idx.txt
```

```
import sys
infile = sys.argv[1]      # samtools idxstats 결과 파일
inopen = open(infile)
total_read = 0
for line in inopen:
       linesp = line.split('\t')
       chr_reads = int(linesp[2]) #idxstats 의 결과에서 염색체 별 mapped read 정보를 저장
       total_read += chr_reads     #저장한 정보를 차례대로 더함
inopen.close()
print total_read #total mapped read 를 출력
```

- 결과물
  - Total mapped read

```
4,530,349
```

4) HTSeq 의 read count 를 RPKM/FPKM 으로 변환

- 내용
  - HTSeq 의 결과로 얻은 RNA-seq 의 read count 를 유전자의 발현량의 한 단위로 쓰이는 RPKM/FPKM 단위로 변환함.
  - RPKM 은 Read Per Killobase per Million mapped reads 의 줄임말로, 1 RPKM 은 대략 세포 안에서 1 개의 RNA 가 있다고 해석됨. RNA-seq 이 single-end 타입일 경우 RPKM 이 구해짐.
  - FPKM 은 RPKM 의 앞 글자인 read 가 fragment 로 바뀐 것으로, 구하는 방법은 RPKM 과 동일함. RNA-seq 이 paired-end 인 경우에 구해짐.

- 명령어의 예 (in-house Python script)

```
python readcount_to_rpkm.py data1_htseq_count.txt geneLength.txt 4530349
/home/name_of_user/
```

```
import sys

count_file = sys.argv[1]     #HTSeq 의 결과물 파일
length_file = sys.argv[2]    #유전자 길이 파일
total_read = int(sys.argv[3])       #total mapped read
outdir = sys.argv[4]

count_dict = {}
length_dict = {}

count_open = open(count_file)
length_open = open(length_file)

for count_line in count_open:
       count_line_sp = count_line.strip('\n').split('\t')
       count_dict[count_line_sp[0]] = int(count_line_sp[1])

count_open.close()

for length_line in length_open:
       length_line_sp = length_line.strip('\n').split('\t')
       length_dict[length_line_sp[0]] = int(length_line_sp[1])
length_open.close()

outfile = outdir + 'rpkm.txt'
outopen = open(outfile, 'w')

for gene in count_dict.keys():
       readPerGene = count_dict[gene]
       geneLen = length_dict[gene]
       rpkm = (1000000000*readPerGene/(float)(total_read))/geneLen
       outopen.write(gene + '\t' + str(rpkm) + '\n')
outopen.close()
```

● 결과물

- geneLength.txt: 각 유전자마다 HTSeq 에 사용된 유전자의 길이가 계산된 파일

```
ENSG00000099942.8       17.5411305645
ENSG00000100243.16      0.0
ENSG00000215012.4       0.4257977356
ENSG00000233010.1       1.90742894161
```

6. 유전자 레벨의 차등발현 분석

● 내용

- DESeq 프로그램을 사용하여 차등 발현되는 유전자들을 확인함.
- DESeq 은 HTseq 프로그램의 결과파일을 입력파일로 활용하여 샘플 간의 technical error 및 variation 을 측정하고 보정하여 차등 발현되는 유전자를 분석함.
- DESeq 이 가장 상용적으로 쓰이지만 최근 연구에서 limma package[10]의 voom 함수가 정확도가 높은 것으로 묘사되고 있음.

1) 차등발현 분석을 위한 전처리 과정

● 내용

- 차등발현 분석을 위해 HTSeq 결과물을 DESeq 분석에 맞게 전처리함
- Replicate 가 존재하는 여러 샘플로부터 통합된 read count 데이터를 생산함

● 파일 준비

- 한 컨디션에서 생성된 HTSeq 결과물과 대응하는 반복 샘플 데이터의 HTSeq 결과물 (예: data1_1.count, data1_r.count)
- 다른 컨디션에서 생성된 HTSeq 결과물과 대응하는 반복 샘플 데이터의 HTSeq 결과물 (예: data2_1.count, data2_r.count)
- HTSeq 결과물들을 합칠 python script

● 명령어의 예 (in-house Python script)

```
python makeDEinput.py /path/to/data/files/ data1_1.count data1_2.count
data2_1.count data2_r.count
```

```
## makeDEinput.py: merge all files by gene id

import sys

data_dir = sys.argv[1]   #데이터 파일의 디렉토리
data_files = sys.argv[2:]         #데이터 파일들의 이름

outfile = data_dir + 'Merged_data.count'  #결과 파일 이름 지정

data_dict = {}

for dfile in data_files:
       dopen = open(data_dir + dfile)
       dlines = dopen.readlines()          #각 데이터 파일을 열어서 내용을 읽음
       dopen.close()
       data_dict[dfile] = {}
       for dline in dlines:
              dlinsp = dline.strip('\n').split('\t')
              gene_id = dlinsp[0] #유전자 아이디를 저장
              count = dlinsp[1]   #유전자에 할당된 read count 를 저장
              data_dict[dfile][gene_id] = count    #유전자 아이디와 read count 를 매치시켜 저장

gene_list = data_dict[dfile].keys()       #전체 유전자 리스트

outopen = open(outfile, 'w')      #결과 파일을 열음
for gene in gene_list:
       outopen.write(gene)        #유전자 아이디 출력
       for dfile in data_files:
              outopen.write('\t' + data_dict[dfile][gene]) #각 데이터 파일에서 얻은 read count 를
차례로 출력
       outopen.write('\n')
outopen.close()  #결과 파일을 닫음
```

- 결과물
  - Merged_data.count: 유전자별로 HTSeq 결과물이 통합된 파일

```
ENSG00000008735.10      5       10      6       12
ENSG00000015475.14      448     896     449     898
ENSG00000025708.8       460     920     461     922
ENSG00000025770.14      341     682     342     684
```

2) DESeq 분석

- 내용: R 프로그램에서 DESeq 패키지를 실행하여 유전자들의 차등 발현을 분석한다.
- 파일 준비
  - 모든 샘플에 대하여 유전자별로 통합된 HTSeq 결과물 (예: Merged_data.count)

● 명령어의 예 (DESeq 프로그램)

```
source('http://bioconductor.org/biocLite.R') #bioconductor 에서 패키지를 불러옴
biocLite('DESeq')      #DESeq 설치
library('DESeq')       #DESeq 패키지 로딩
ctable = read.delim('Merged_data.txt', header=F, row.names=1) #read count 파일을 불러옴
colnames(ctable) = c('wt1', 'mut1', 'wt2', 'mut2')  #파일의 열들의 이름 지정
conds = factor(c('wt', 'mut', 'wt', 'mut'))  #파일의 열 별로 컨디션 명시
cds = newCountDataSet(ctable, conds) #컨디션 정보를 포함한 데이터 셋으로 저장
cds = estimateSizeFactors(cds)        #샘플간에 총 read 개수를 보정
cds = estimateDispersions(cds)        #샘플간에 분산 및 오류를 예측
res = nbinomTest(cds, 'wt', 'mut')    #샘플간 차등 발현 측정
resSig = res[res$padj<0.05]   #보정된 p-value 가 0.05 인 것만 추출
write.csv(res, file='all_data.csv')  #전체 데이터 파일에 쓰기
write.csv(resSig, file='sig_data.csv')       #p-value 로 뽑은 데이터만 파일에 쓰기
```

● 결과물

- all_data.csv: 모든 유전자의 차등발현 분석결과와 발현 비율이 계산된 파일
- sig_data.csv: *p*-value 가 0.05 이하로 차등발현 분석결과가 유의미하게 나타나는 유전자의 결과가 정리된 파일

```
id   baseMean  baseMeanA  baseMeanB foldChange log2FoldChange pval padj

1 ENSG00000008735.10   7.771554   7.771554   7.771554         1  -3.203427e-16 1    1    1
2 ENSG00000015475.14 634.298872 634.298872 634.298872         1  -3.203427e-16 2    1    1
```

7. 참조 전사체에 RNA-seq 데이터 alignment

● 내용

- 유전자가 아닌 이소체 레벨로 발현량 차이를 분석하기 위해 품질 관리된 시퀀싱 데이터를 bowtie 로 참조 전사체에 맵핑.

● 파일 준비

- FASTQ 포맷의 품질 관리된 시퀀싱 데이터 파일 (예: data1_1.trimmed.fastq)
- FASTA 포맷의 참조 전사체 데이터 파일 (예: gencode.v19.annotation.fa)

1) 참조 전사체 데이터 파일 다운로드

● 내용: 시퀀싱 데이터를 맵핑할 참조 전사체 데이터 파일의 다운로드.

● 방법

- GENCODE[14] 데이터 다운로드 페이지로 이동.
- 해당 생물 종의 참조 유전체 버전에 대응하는 참조 전사체 데이터 선택 (버전 19: 인간 유전체 버전 GRCh37 에 대응, 버전 25: 인간 유전체 버전 GRCh38 에 대응, 버전 M1: 마우스 유전체

버전 GRCm37 에 대응, 버전 M10: 마우스 유전체 버전 GRCm38 에 대응).

- FASTA 포맷의 "Protein-coding transcript sequences" 와 "Long non-coding RNA transcript sequences" 파일을 원하는 위치에 다운로드.
- 다운로드 받은 "Protein-coding transcript sequences" 와 "Long non-coding RNA transcript sequences" 파일을 단일 파일로 만듦.

● 명령어의 예 (Linux 기본 명령어)

```
cat gencode.v19.pc_transcripts.fa gencode.v19.lncRNA_transcripts.fa >
gencode.v19.transcripts.fa
```

● 결과물

- gencode.v19.transcripts.fa: 전사체의 시퀀스 정보가 합쳐진 FASTA 포맷 파일

```
>19 chr1:24480647-24513765 NM_173064
TCAATTGTACCTTTTTTCTTTAAAAATAGAGCTCTTTATTTAAACAGTTTAGGGTAATAC
CAACAAGCAGTAGGATATCGTTTAAATATGACATAAACATAGAAAAAATAGAAATAAATA
```

2) 참조 전사체 데이터 파일의 인덱스 생성

● 내용: 맵핑에 사용할 참조 전사체 데이터 파일의 인덱스 생성.

● 명령어의 예 (Bowtie2 프로그램)

```
bowtie2-build -f gencode.v19.transcripts.fa gencode.v19.transcripts
```

● 옵션의 설명

-f: 참조 전사체 서열 데이터 파일 (FASTA 포맷)

● 결과물

- 참조 전사체 서열 데이터의 인덱스인 .bt2 파일

3) 참조 전사체에 시퀀싱 데이터 맵핑

● 내용

- Bowtie2 로 품질 관리된 시퀀싱 데이터를 참조 전사체 데이터에 맵핑.
- Bowtie2 는 짧은 길이의 read 로 이루어진 시퀀싱 데이터를 빠르게 맵핑함.
- 인덱스 파일은 위 과정 중에서 bowtie2-build 로 만든 .bt2 확장자 파일을 얘기하며, 맵핑 명령어 입력시 확장자를 제외하고 파일명만 입력함 (예: /인덱스_파일_디렉토리/gencode.v19.transcripts).
- 시퀀싱 데이터가 paired-end 인 경우, 공백으로 첫 번째 read 입력 파일과 두 번째 read 입력 파일을 구분함.
- Bowtie2 의 결과물로 SAM 포맷의 맵핑된 데이터를 얻음.

- 명령어의 예 (Bowtie2 프로그램)

```
bowtie2 -q --fr -k 100 --no-mixed --no-discordant -x /인덱스_파일_디렉토리
/gencode.v19.transcripts -1 data1_1.trimmed.fastq -2
data1_2.trimmed.fastq -S data1.mapped.sam
```

- 옵션의 설명

-q: 입력 데이터 파일의 종류 (-q: FASTQ, --qseq: Illumina's qseq, -f: FASTA)

--fr: 해당 시퀀싱 데이터의 라이브러리 타입 (--fr: Forward-reverse, --rf: Reverse-forward, --ff: Forward-forward, 입력 데이터의 타입이 paired-end 인 경우에만 해당됨)

-k: 결과물 (SAM 포맷의 맵핑된 데이터)에 기록 가능한 각 read 별 최대 맵핑 허용치

--no-mixed: 입력 파일의 첫 번째 read 와 두 번째 read 가 함께 맵핑되지 않는 결과는 출력하지 않음 (입력데이터의 타입이 paired-end 인 경우에만 해당됨)

--no-discordant: 입력 파일의 read 가 비정상적으로 맵핑 되는 결과는 출력하지 않음 (read 의 방향, read 간의 거리가 예상과 다른 경우)

-x: 맵핑에 사용할 참조 전사체 인덱스 파일 (디렉토리 포함)

-1: 품질 관리된 입력 데이터의 첫 번째 read 파일 (입력 데이터의 타입이 paired-end 인 경우에만 해당됨, 입력 데이터의 타입이 single-end 인 경우에 -U 옵션으로 대체함)

-2: 품질 관리된 입력 데이터의 두 번째 read 파일 (입력 데이터의 타입이 paired-end 인 경우에만 해당됨)

-S: SAM 포맷의 맵핑된 결과물을 생성하고 저장할 파일 이름

- 결과물

- data1.mapped.sam: 맵핑된 결과물이 SAM 포맷으로 저장된 파일 [15)]

i. read name
ii. flag information
iii. reference sequence name
iv. 1-based leftmost mapping position
v. mapping quality
vi. CIGAR tag
vii. reference name of the mate read
viii. position of the mate read
ix. observed template length
x. segment sequence
xi. ASCII of Phred-scaled base quality

```
HWI-ST610R:586:C2K7GACXX:2:1101:1599:25393     83     4424    1073    1      101M    =      975     -199
GGGTATAGATGACCTGGAAACCATGCCAGATGACCTTCTGACCACGTTGGATGACACTTGTGATCTCTTTGCCCCCCTAGTCCAGGAGACTAATAAGCAGC
DDDCDACCCA>CDDDDDDCCDDDDCCCCCDAA>DC@CDDCBBA8@CDDDDEDDDC@DCDCADDDDDDB?6DDJIIJHFGHHIHHGHGFFHGHHDFDFF@@@
AS:i:0  XS:i:0  XN:i:0  XM:i:0  XO:i:0  XG:i:0  NM:i:0  MD:Z:101       YS:i:0  YT:Z:CP
```

4) 맵핑의 품질 검사

- 내용:
  - 전사체에 맵핑을 끝낸 후 생성된 BAM 또는 SAM 파일을 사용하여 맵핑 상태의 기초 통계를 알아본다.
  - 내용은 **4. 참조 유전체 데이터에 RNA-seq data alignment** 의 **맵핑 품질 검사**를 참조한다.

8. 이소체 발현량 측정

1) 발현량 측정을 위한 전처리

- 내용:
  - 전사물의 발현량 측정에는 BitSeq 을 이용함.
  - BitSeq 은 RNA-seq 데이터로 만든 사후확률[16]에서 MCMC 샘플링[17]을 통해 상대적인 발현량을 계산하는 프로그램임.
  - 주로 반복 실험 데이터를 가지고, 기술적 오류와 생물학적 오류를 모두 고려하기 때문에 보다 정확한 발현량 예측이 가능함.
  - BitSeq 을 이용하여, 앞의 단계에서 mapping 이 끝난 SAM 파일을 가지고 read 가 각 전사물에 mapping 될 확률을 구함.

- 명령어의 예 (BitSeq 프로그램)

```
/BitSeq_설치_디렉토리/BitSeq/parseAlignment data1.mapped.sam -o
data1.mapped.prob --trSeqFile gencode.v19.transcripts.fa --trInfoFile
gencode.v19.transcripts.tr --uniform --verbose
```

- 옵션의 설명

  -o: 확률값이 계산된 결과물을 생성하고 저장할 파일 이름

  --trSeqFile: 맵핑에 사용된 참조전사체 데이터 인덱스 생성에 사용한 파일 (FASTA 포맷)
  (예: gencode.v19.transcripts.fa)

  --trInfoFile: 맵핑에 사용된 참조 전사체 데이터 파일의 정보를 생성하고 저장할 파일 이름
  (유전자와 전사체의 이름, 길이 등의 정보를 포함하고 있음)

  --unform: 맵핑된 read 의 uniform distribution 을 가정함 (non-uniform distribution 을 가정할 시 해당 옵션을 포함하지 않음)

  --verbose: 전처리 과정의 정보를 출력함

- 결과물
  - data1.mapped.prob: 각 전사체에 맵핑된 reads 의 통계 결과 및 확률값 (로그 형태)
    i. Ntotal: total number of reads
    ii. Nmap: mapped number of reads
    iii. M: number of transcripts
    iv. Column names: read_name, number_of_alignments, transcript_id and log-scale probability of alignments

```
# Ntotal 2444
# Nmap 2348
# M 5887
# LOGFORMAT (probabilities saved on log scale.)
# r_name num_alignments (tr_id prob )^*{num_alignments}
HWI-ST610R:586:C2K7GACXX:2:1101:1314:82498 7 4698 -1.876376235e+01 4701
-1.874320036e+01 4697 -1.876541116e+01 4700 -1.887060692e+01 4695 -
1.891696964e+01 4699 -1.909503486e+01 0 -8.055107290e+01
```

2)

이소체 레벨의 발현량 측정

- 내용
  - BitSeq 은 이소체 레벨의 발현량 측정을 위한 알고리즘으로 MCMC sampling 과 Variational Bayes (VB)를 제공함.
  - VB 알고리즘은 MCMC sampling 과 비교하여 매우 빠르다는 장점이 있지만 차등 발현 분석에는 적합하지 않음.
  - MCMC sampling 알고리즘은 여러 번의 샘플링을 통해 발현량을 나타내는 확률의 분산을 수렴시키고, 이를 이용하여 이소체의 상대적인 양을 예측함.

- 명령어의 예 (BitSeq 프로그램)

```
/BitSeq_설치_디렉토리/BitSeq/estimateExpression data1.mapped.prob -o
data1 --outType RPKM -p /BitSeq_설치_디렉토리/BitSeq/parameters1.txt -t
gencode.v19.transcripts.tr -P 2
```

옵션의 설명

-o: 이소체 레벨의 발현량이 측정된 결과물을 생성하고 저장할 파일 이름

--outType: 결과물에 저장할 발현량의 타입 (theta, RPKM, counts)

-p: 발현량 측정을 위해 미리 계산된 파라미터가 저장된 파일 (BitSeq_설치_디렉토리에 위치해 있음)

-t: 전처리 과정에서 생성된 참조 전사체 정보 파일 (예: gencode.v19.transcripts.tr)

-P: 발현량 측정에 사용할 thread 개수

- 결과물
  - data1.thetaMeans: 전사체마다 전체적인 발현량, 전체적인 read count, 전체적인 분산, 각 샘플링에서의 발현량을 나타낸 파일

```
# T => Mrows
# M 5887
# file containing the mean value of theta - relative abundace of
fragments and counts
# (overall mean, overall counts, mean of saved samples, and mean from
every chain are reported)
# columns:
# <transcriptID> <meanThetaOverall> <meanReadCountOverall>
<meanThetaSaved> <varThetaOverall> <chain1mean> <chain2mean>
<chain3mean> <chain4mean>
#thetaAct: 0.000000000e+00 0 -inf 0.000000000e+00 0.000000000e+00
0.000000000e+00 0.000000000e+00 0.000000000e+00
1 1.243990626e-04 0 -9.580342996e+00 1.690214146e-08 1.190094733e-04
1.258719444e-04 1.206176051e-04 1.320972276e-04
```

- data1.rpkm: 전사체마다 500 개의 MCMC 샘플에서의 발현량 (RPKM)이 쓰여진 파일

```
1.532317616e+02 1.037320110e+02 1.381847638e+02 1.205779272e+02
9.780934966e+01 1.946361451e+01 1.452246787e+01 3.760407936e+02
```

9. 이소체 레벨의 차등발현 분석

- 내용
  - BitSeq 을 이용하여 서로 다른 컨디션으로 생성된 시퀀싱 데이터로부터 차등 발현되는 이소체 분석을 수행함.
  - MCMC sampling 알고리즘으로 측정된 발현량을 입력데이터로 사용함 (예: data1.rpkm).
  - 반복 시퀀싱 데이터가 존재하면 정확한 차등 발현 분석을 수행하는데 도움을 줄 수 있음.

- 파일 준비
  - 하나의 컨디션에서 반복 시퀀싱된 데이터로부터 발현량을 측정한 파일 (예: data1-1.rpkm, data1-2.rpkm)
  - 다른 컨디션에서 반복 시퀀싱된 데이터로부터 발현량을 측정한 파일 (예: data2-1.rpkm, data2-2.rpkm)

1) 발현량의 평균 및 분산 계산

- 내용: 반복 시퀀싱 데이터로부터 분산을 구하여 technical bias 를 보정하고, 컨디션 별로 유전자/이소체의 평균 발현량과 분산을 계산하여 각 이소체의 발현 패턴을 분석함.

- 명령어의 예 (BitSeq 프로그램)

```
/BitSeq_설치_디렉토리/BitSeq/getVariance --log -o data.Lmean data[12]-
```

- 옵션의 설명

  --log: 발현량을 로그 값으로 계산함 (BitSeq 은 로그 값의 발현량을 사용함)

  -o: 이소체의 평균 발현량과 분산을 계산한 결과물을 생성하고 저장할 파일 이름

  - 각 컨디션 별로 반복 시퀀싱된 데이터로부터 발현량을 측정한 파일을 모두 동시에 입력 데이터로 받음.

- 결과물

  - data.Lmean: 각 전사체의 평균 발현량과 분산을 log-scale 로 나타낸 결과 파일

```
# Transcripts mean expression and sample variance.
# files: sample9_est.rpkm sample63_est.rpkm
# L -> values logged
# M 5887
4.405283872e+00 1.550256895e+00
5.327010792e+00 1.577683949e+00
```

2) Hyperparameter 구하기

- 내용
  - 평균 발현량의 신뢰도를 보정하기 위해, 발현량에 따라 달라지는 hyperparameter 를 계산함.
  - Hyperparameter 는 발현량의 계산을 위해 쓰이는 사전확률[18]의 분포를 결정하는 요소임
  - Hyperparameter 는 MCMC 샘플링을 통해 발현량이 유사한 유전자 그룹별로 하나씩 예측됨.
- 명령어의 예 (BitSeq 프로그램)

```
/BitSeq_설치_디렉토리/BitSeq/estimateHyperPar --meanFile data.Lmean -o
data.param data1-[12].rpkm C data2-[12].rpkm
```

- 옵션의 설명

  --meanFile: 평균 발현량 및 분산 계산을 통해 생성된 결과물을 입력파일로 받음 (예: data.Lmean)

  -o: 발현량에 따른 hyperparameter 계산값을 생성하고 저장할 파일 이름

  - 반복 시퀀싱된 데이터로부터 발현량을 측정한 파일을 컨디션 별로 구분하여 입력 데이터로 받음 (글자 "C"를 이용하여 컨디션이 다른 데이터들을 구분함).

- 결과물
  - data.param: 계산된 hyperparameter (alpha, beta) 값이 쓰여진 파일

```
# lambda0 2
# alphaSmooth f: 0.2 nSteps: 5
# betaSmooth f: 0.2 nSteps: 5
# PN 110 hyperparameters
# columns: alpha beta expression
627.546 632.773 -0.817936
783.371 755.171 -0.169072
866.099 824.784 0.179002
```

3) 특정 컨디션 발현 및 PPLR[19] (Probability of Positive Log Ratio) 구하기

- 내용
  - 각각의 이소체가 특정 컨디션에서 평균 발현량이 다를 확률 (PPLR)를 계산함.
  - PPLR 은 각각의 이소체가 MCMC sampling 을 했을 때 첫 번째 컨디션에서 발현량이 더 컸던 경우의 비율을 말함.
  - 따라서 이소체가 첫 번째 컨디션에서의 평균 발현량이 클 확률은 PPLR 이 0 에 가까울수록 줄어들며, 1 에 가까울수록 커짐.
  - PPLR 의 값을 가지고 차등 발현되는 이소체의 순위를 나타낼 수 있음
- 명령어의 예 (BitSeq 프로그램)

```
/BitSeq_설치_디렉토리/BitSeq/estimateDE -o data -p data.param data1-
[12].rpkm C data2-[12].rpkm
```

- 옵션의 설명

  -o: 차등 발현 분석 결과를 생성하고 저장할 파일 이름

  -p: 발현량에 따른 hyperparameter 계산을 통해 생성된 결과물을 입력 파일로 받음

  - 반복 시퀀싱된 데이터로부터 발현량을 측정한 파일을 컨디션 별로 구분하여 입력 데이터로 받음 (C 로 다른 컨디션의 데이터를 구분함)

- 결과물

  - data.pplr: PPLR, mean log2 fold change, confidence intervals 와 mean condition mean expression 결과가 포함돼 있음.

```
# data1_[12].rpkm C data2_[12].rpkm
# lambda_0 2
# T
# M 5887
# N 500
# Conditions: C 2 Condition pairs(1): 1~2
# Columns contain PPLR for each pair of conditions, log2 fold change
with confidence intervals for each pair of conditions and log mean
condition mean expression for each condition.
# CPxPPLR CPx(log2FC ConfidenceLow ConfidenceHigh) Cx(log mean condition
mean expressions)
0.526 0.0522545 -3.10767 2.97767 4.33924 4.37546
0.486 -0.1016 -3.52761 2.94988 5.28634 5.21592
```

10. Reference

1) http://www.bioinformatics.babraham.ac.uk/projects/fastqc/
2) https://github.com/najoshi/sickle
3) https://github.com/alexdobin/STAR
4) http://rseqc.sourceforge.net/#bam-stat-py
5) https://www.python.org/downloads/
6) https://pypi.python.org/pypi/HTSeq
7) https://www.r-project.org/
8) http://bioconductor.org/packages/release/bioc/html/DESeq.html
9) http://bowtie-bio.sourceforge.net/bowtie2/index.shtml
10) http://bioconductor.org/packages/release/bioc/html/limma.html
11) http://bitseq.github.io/
12) http://samtools.sourceforge.net/
13) ftp://ftp.ebi.ac.uk/pub/databases/blueprint/reference/
14) http://www.gencodegenes.org/
15) https://broadinstitute.github.io/picard/explain-flags.html
16) https://en.wikipedia.org/wiki/Posterior_probability
17) https://en.wikipedia.org/wiki/Markov_chain_Monte_Carlo

18) https://en.wikipedia.org/wiki/Prior_probability
19) https://github.com/BitSeq/BitSeq/wiki/PPLR