sRNA-seq 표준 분석 프로토콜 (SOP)

2016.10.14

한양대학교 생명과학과
남진우 교수

해당 sRNA-seq Standard Operation Protocol (SOP)은 sRNA-seq 을 이용한 기본적인 microRNA 분석 프로토콜을 목적으로 하여 설계되었음. 여기서 다루고 있는 sRNA-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 (sRNA-seq SOP 는 2016.10.14 현재 기준으로 작성되었음.).

목 차

1. 배경

1) 연구 배경

miRNA는 약 22 nucleotide 길이를 갖는 RNA로, seed region (nucleotide 2-7)이 mRNA의 3′UTR 상에 존재하는 target site와 complementary base pairing을 통해 target mRNA의 발현을 억제하는 것이 대표적인 역할로 알려져 있음. 이로 인해, miRNA의 발현량은 taget mRNA의 발현량과 negative correlation을 가지게 되므로, miRNA의 발현량을 분석하는 것은 mRNA의 발현 패턴을 이해하는 중요한 요소가 될 수 있음. 더욱이, miRNA는 조직 및 세포 특이적으로 발현되는 특징이 있기 때문에, sRNA-seq을 통해 miRNA 발현량을 정확히 분석하는 것은 다양한 질병 연구에서 시도 되고 있음. 이로 인해 miRNA의 발현량을 분석하는 다양한 파이프라인들이 개발되었지만 miRNA의 5′ heterogeneity, arm-switching, 3′ end mismatch 등의 생물학적 특징을 통합적으로 고려한 파이프라인이 존재하지 않아 정확한 miRNA 발현 패턴을 분석하는데 어려움이 존재함. 본 SOP (Standard Operation Procedure)을 통해 small RNA-seq 데이터를 바탕으로 miRNA의 정확한 발현량뿐만 아니라 miRNA의 생물학적인 변이를 이해할 수 있는 데이터를 제공하는 표준 파이프라인을 제시하고자 함.

2) miRNA 발현량 분석 파이프라인

small RNA-seq 데이터를 이용해서 miRNA 발현량을 분석하는 전체 과정은 그림 1과 같음. miRNA 발현량 분석 파이프라인은 크게 전처리 과정과 발현량 분석 과정으로 이루어짐. 전처리 과정에서 기본적으로 원시 데이터의 quality를 확인하는 작업을 거친 후, small RNA-seq read들에 존재하는 adapter sequence를 제거함. Trim된 read들을 분석 과정의 효율화를 위해 collapse함으로써 전 처리가 완료됨. 전처리 과정을 거친 read들을 이용해서 annotation되어 있는 miRNA sequence에 맵핑하는 과정을 통해서 결과물을 만듦. miRNA의 생물학적 특성을 고려해 mapped reads의 quality를 확인한 후, 정확하게 mature miRNA에 맵핑된 발현량, miRNA의 5′ heterogeneity, arm-swithcing과 3′ end mismatch 등을 확인할 수 있는 데이터를 생산하는 최종 과정을 거침.

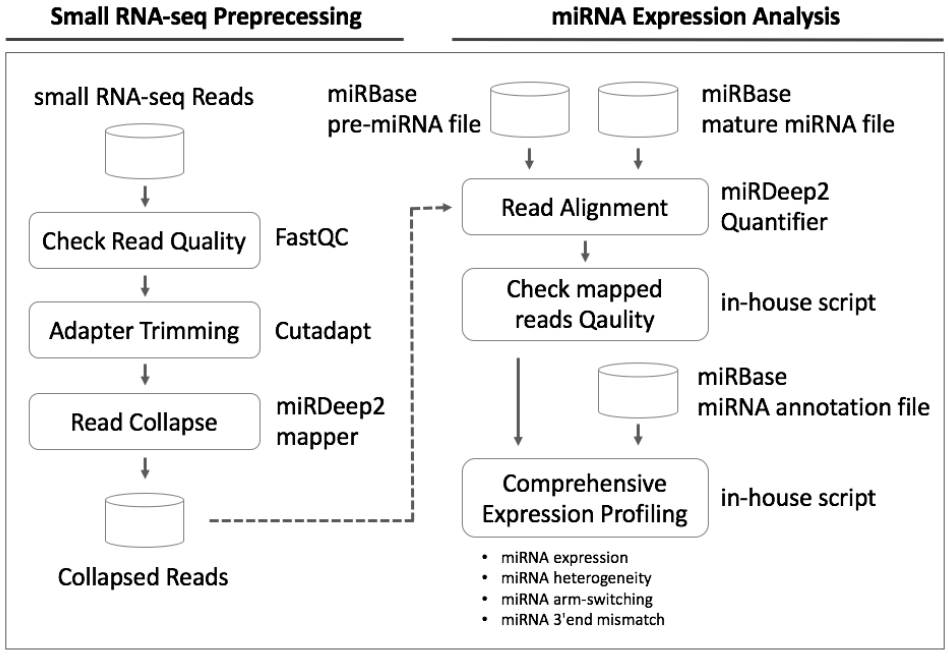

그림 1. miRNA 발현량 분석 과정 모식도

2. 준비

1) 원시 데이터

- Read length: 1x50nt (single read)
- Sequencing depth: > 1G
- Library type: strand specific RNA Library
- Sequencing platform: illumine

2) 사용 프로그램

- FastQC 설치[1)]
- Cutadapt 설치[2)]
- Perl 및 perl package (Getopt::Std, File::Copy, File::Path, Term::ANSIColor) 설치[3)]
- miRDeep2 설치[4)]

3. 원시 데이터 및 품질 관리

1) FastQC로 원시 데이터 품질 확인

- 내용: FastQC는 데이터에 포함되어 있는 read 들의 base quality, 길이 분포, GC content, 반복 서열 등을 통계적으로 보여주며, 이를 이용해 원시 데이터의 품질을 확인.

- 명령어의 예 (FastQC 프로그램)

```
fastqc –o ~/OUTPUT_DIRECTORY –f fastq –t 4 –q FASTQ_FILE1.fastq
FASTQ_FILE2.fastq
```

- 옵션의 설명

  -f: 적용할 데이터의 형태

  -q: 적용할 데이터

  -t: FastQC에 사용할 thread 수

  -o: 결과를 저장할 directory

- 결과물

  - Fastqc_data.txt: read의 질과 길이 분포, GC content, 반복 서열 등이 기술되어 있는 파일.
  - Fastqc_report.html: 글자로 기술되어 있는 위의 파일을 그래프와 표를 통해 보여주는 파일.
  - Fastqc manual: http://dnacore.missouri.edu/PDF/FastQC_Manual.pdf

2) Cutadapt로 원시 데이터 adapter 제거

- 내용: FastQC로 adapter sequence와 read length distribution을 확인 후, cutadapt를 이용한 adapter trimming과 length filtering 진행. 최근의 sRNA sequencing data는 대부분 5′ adapter sequence가 제거된 상태로 제공이 되므로 3′ adapter sequence만을 제거함. 또한, miRNA의 길이는 평균 22 nucleotide로, DROSHA와 DICER에 의한 processing의 오류나 mutation을 고려해 18~26 nucleotide 길이를 가진 read만 걸러냄.

- 기준
  - Adapter sequence overlap: 6
  - Read 길이: 18 nucleotide 이상, 26 nucleotide 이하

- 명령어의 예 (Cutadapt 프로그램)

```
cutadapt --overlap=6 -f fastq -a TGGAATTCTCGGGTGCCAAGG -m 18 -M 26 -o
~/outputDir/OUTPUT_FILE ~/inputDir/INPUT_FILE
```

- 옵션의 설명

  --overlap: read와 adapter sequence의 최소 overlap 기준 설정 (e.g.) 3′ adapter trimming시, --overlap=6을 주면, adapter의 5′ end 6 nucleotide 를 read의 3′ end에서 5′ end 방향으로 같은 sequence를 검색해 trimming)

  -a: 3′ adapter sequence

  -m: LENGTH 보다 짧은 read 버림

  -M: LENGTH 보다 긴 read 버림

  -o: 결과물 위치 및 이름 지정

- 결과물
  - Fastq format의 파일 생성: adapter sequence가 제거되고 길이가 16~26인 read들만 존재.
  - Cutadapt manual: https://media.readthedocs.org/pdf/cutadapt/stable/cutadapt.pdf

4. miRNA 발현량 프로파일링

1) miRDeep2 mapper을 이용하여 miRNA 데이터 alignment

- 내용: 같은 sequence를 가진 read들을 병합하여 fasta 파일을 생성한 후 이를 이용하여 bowtie 맵핑을 진행하여 .arf 파일을 생성함. 해당 aligned reads 파일은 추후에 novel miRNA 분석을 위해서 쓰일 수 있음. 하지만 발현량 프로파일링을 위한 파이프라인에서는 quantifier 과정에서 필요한 fasta 파일만 생성함.

- 명령어의 예 (miRDeep2 mapper 프로그램)

```
perl mapper.pl INPUT_FILE.fastq -e -h -j -m -s OUTPUT_FILE.fa
```

- 옵션의 설명

  -e: 적용할 데이터의 형태가 fastq임을 명시

  -h: 결과물을 fasta 형태로 생성

  -j: a, c, g, t, u, n, A, C, G, T, U, N을 제외한 sequence를 제거

  -m: 같은 sequence를 가진 read들을 병합

  -s: 병합한 read들을 이 결과물에 출력

- 결과물
  - Fasta format의 파일 생성

```
>seq_0_x347961      (seq_0 이라는 read 가 347,961 개 존재)
AAGCTGCCAGTTGAAGAACTGT  (seq_0 의 sequence
```

2) miRDeep2 Quantifier을 이용하여 mature miRNA의 발현량 추정 파일 생성

- 내용: mapper의 결과물인 병합된 read들을 precursor miRNA에 bowtie를 이용해서 맵핑을 진행. Default 옵션으로 'bowtie -f -v 1 -a --best --strata –norc'가 사용됨.

  -f: input file의 형태가 fasta라는 것을 지정
  -v 1: read의 mismatch를 한 개 허용
  -a: align된 모든 reads를 report함
  --best: 각 read의 alignment들을 mismatch가 적은 순으로 report.
  --strata: 각 read의 alignment들 중 mismatch가 가장 적은 alignment들을 report
  --norc: reverse complement reference sequence에 맵핑 하지 않음

  precursor miRNA 에 맵핑 함과 동시에, known mature miRNA 들을 bowtie 를 이용해서 '–f –v 0 –a --best --strata --norc'의 옵션으로 known precursor miRNA 에 맵핑을 진행함. 그 후, bowtie 결과물들을 통합해서 precursor miRNA 에서 발현되는 mature miRNA 들과 그 mature miRNA 의 위치에 맵핑된 read 들을 함께 보여줌. Input file 로는 mapper 과정으로부터 만들어진 fasta 파일 뿐만 아니라, miRbase 에서 제공하는 hairpin 파일 (특정 종의 precursor miRNA sequence file)과 miRbase 에서 제공하는 mature 파일 (특정 종의 mature miRNA sequence file)이 필요함.

- 명령어의 예 (miRDeep2 Quantifier 프로그램)

```
perl quantifier.pl –p HAIRPIN_FILE.fa –m MATURE_FILE.fa –r
COLLAPSED_READS_FILE.fa –t hsa –g 2 –e 2 –f 5
```

- 옵션의 설명

  -p: mirbase_hairpin file 지정
  -m: mirbase_mature file 지정
  -r: mapper 결과 file 지정
  -t: 종 지정
  -g: read들을 precursor에 맵핑할 때 허용할 mismatch의 수
  -e: read들을 precursor에 맵핑할 때 허용할 mature miRNA upstream nucleotide 수
  (ex) –e 2: mature miRNA보다 2 nucleotide upstream까지 맵핑되는 것을 허용)
  -f: read들을 precursor에 맵핑할 때 허용할 mature miRNA downstream nucleotide 수
  (e.g.) –f 5: mature miRNA보다 5 nucleotide downstream까지 맵핑 되는 것을 허용)

- 결과물
  - miRBase.mrd: precursor miRNA마다 전체 맵핑된 read의 수, 해당되는 mature miRNA들에 맵핑된 read의 수, precursor상에서 mature miRNA의 위치, 그 부분에 맵핑된 read의 sequence를 보여주는 결과 파일

```
>hsa-let-7a-1
total read count            91396
hsa-let-7a-5p read count    91162
hsa-let-7a-3p read count    234
remaining read count        0
exp                fffff5555555555555555555555ffffffffffffffffffffffffffffff333333333333333333333fff
pri_seq            ugggaugagguaguagguuguauaguuuuagggucacacccaccacugggagauaacuauacaaucuacugucuuuccua
pri_struct         (((((.(((((((((((((((((((.....(((...((((....)))).)))))))))))))))))))))))))))))
#MM
seq_5632585_x1     ...UCugagguaguagguuguauagu.....................................................
2
seq_5472391_x2     ...UCugagguaguagguuguauaguu....................................................
2
seq_5986713_x1     ....Cugagguaguagguuguauag......................................................
1
```

3) 생물학적 측면에서 sequencing 질 확인
   - 내용: 실제 동물 세포에서 발현되는 대부분의 miRNA는 22 nucleotide 길이를 가지며, 5′ terminal의 uracil이 많이 나타나는 bias가 존재함. 이러한 생물학적 지식을 바탕으로 sequencing data의 질을 확인. mismatch가 없이 맵핑된 read들 중에서, 길이와 5′ end nucleotide에 따른 read들의 분포를 파악.

- 명령어의 예 (in-house script: mirdeepCheckSequencingQuality.py)

```
python    mirdeepCheckSequencingQuality.py    ~/mirDeepOutDir/mapped.arf
~/outputDir/outputFile.txt
```

Input #1: miRDeep2의 quantifier에서 리드들을 precursor miRNA에 맵핑하여 생성된 arf 형식 파일을 입력.

Input #2: 결과를 저장할 파일을 입력.

```
def check_quality(inputF):
    qualityDic = dict()
    for i in xrange(18,27):  #miRNA candidate 들의 길이
        qualityDic[i] = {'T':0,'G':0,'C':0,'A':0}     #가능한 첫
nucleotide 종류
    f = open(inputF); lines = f.readlines(); f.close()
    for line in lines:  #collapse 된 read 들의 매핑 정보를 가진 줄
        line = line.strip('\n').split('\t')
        readName = line[0].split('_x')[0]   #collapse 된 read 의 이름
        readNumber = int(line[0].split('_x')[1]) #collapse 된 read 의 수
        readLength = int(line[1])  #read 의 길이 정보
        readSequence = line[4].upper()
        readMatch = line[12]  #read 의 match / mismatch 정보
        if readMatch.find('M')>=0: continue #mismatch 없이 맵핑된 read 들만
봄
        qualityDic[readLength][readSequence[0]] += readNumber       #첫
nucleotide 따라 맵핑된 read 를 나눠 collapse 된 read 의 개수를 더함
    return qualityDic

def write_qualityDic(qualityDic, outputF):
    writer = open(outputF, 'w')
    header = 'Sequencing_quality' + '\t' + 'T' + '\t' + 'C' + '\t' +
'G' + '\t' + 'A'
    writer.write(header + '\n')
    for readLength in qualityDic.keys():
        writer.write(str(readLength) + '\t')
        writer.write(str(qualityDic[readLength]['T']) + '\t')
        writer.write(str(qualityDic[readLength]['C']) + '\t')
        writer.write(str(qualityDic[readLength]['G']) + '\t')
        writer.write(str(qualityDic[readLength]['A']) + '\n')
    writer.close()

def main(inputF, outputF):
    qualityDic = check_quality(inputF) #첫 nucleotide 에 따라서 맵핑된 read
를 나눠 총 read 개수 정보를 저장하는 dictionary 생성
    write_qualityDic(qualityDic, outputF)  #dictionary 의 정보를 저장

if __name__ == '__main__':
    import sys
    inputF = sys.argv[1]     #collapsed reads 를 맵핑한 결과인 arf 파일
    outputF = sys.argv[2]    #결과물을 저장할 파일
    main(inputF, outputF)
```

(수정 추가됨)

- 결과물

  - Read의 길이 (18 ~ 26 nucleotide) 당, 5′ end nucleotide (T, C, G, A) 당 read의 개수를 기록한 파일

```
Sequencing_quality     T      C      G      A
18                   6102   1001    606    2901
19                   15258  2030    3214   3880
20                   226442 22427   2611    11905
21                   288915 344746  8961    57595
22                   1209051 50001   6183    448856
23                   290479 43042   2805    70583
24                   117180 6982    319     4739
25                   264    173     91      1258
26                   92     65      61      385
```

4) miRNA 발현량 보정

- 내용: quantifier에서 bowtie 맵핑 시, reads의 multi loci 맵핑을 허용하기 때문에, miRBase.mrd 파일에 하나의 read가 여러 mature miRNA에 반복해서 맵핑 되어 기록됨. miRNA 발현량 프로파일에서 중복 맵핑에 의한 bias를 최소화하기 위해 mature miRNA에 중복 맵핑된 read들을 해당 precursor miRNA에 맵핑된 전체 read count의 비율로 나누는 작업을 진행 함.

- 명령어의 예 (in-house script: miRDeep2_revise_mrd.py)

```
python miRDeep2_revise_mrd.py ~/mirDeepOutDir/miRBase.mrd
```

  - Input #1: miRDeep2의 quantifier를 이용해서 만들어진 결과물인 miRBase.mrd 파일을 입력

```
def parsing(mrdFile):
    mrdDic = dict()
    file = open(mrdFile); all = file.read(); file.close()
    chunks = all.split('>')[1:] #precursor miRNA 별로 분리
    for chunk in chunks:
        lines = chunk.split('\n')
        pre_miRNA = lines[0] # precurcor miRNA의 이름.
        total_read_count = lines[1].split(' ')[-1]    #pre_miRNA에 맵핑된 전체
read 수
        seq_lines = filter(lambda x: 'seq_' in x, lines)    #맵핑된 read들
        if len(seq_lines) == 0: continue    #맵핑된 read가 있는 pre_miRNA만 봄
        for seq_line in seq_lines:
            seq = seq_line.split(' ')[0]    #read 이름
            if not mrdDic.has_key(seq): mrdDic[seq] = []
            was_line = [pre_miRNA, total_read_count]
            mrdDic[seq].append(was_line)    #read마다 맵핑된 pre_miRNA 정보 저장
    return mrdDic

def comparing(mrdDic):
    newDic = dict()
    for seqID in mrdDic.keys():
        if len(mrdDic[seqID]) == 1: continue    #중복 맵핑된 read들만 봄
```

(수정 추가됨)

```
        total = sum(map(lambda x: int(x[1]), mrdDic[seqID]))    #맵핑된 pre_miRNA
들의 전체 read 수의 합
        for was_line in mrdDic[seqID]:
            if not newDic.has_key(was_line[0]): newDic[was_line[0]] = []
            seq_count = int(seqID.split('x')[1]) #중복 맵핑된 read의 수
            new_line = [seqID, seqID.split('x')[0] + 'x' +
str(int(round(seq_count*int(was_line[1])/float(total))))] #pre_miRNA에 맵핑된
read 수의 비율로 나눠준 read의 수
            newDic[was_line[0]].append(new_line) #pre_miRNA마다 수정 후의 read를 저
장
    return newDic

def revising(mrdFile, newDic):
    writer = open(mrdFile.strip('.mrd') + '_new.mrd','w')
    f = open(mrdFile); all = f.read(); f.close()
    chunks = all.split('>')[1:]
    for pre_miRNA in newDic.keys():
        beRevisedChunk = filter(lambda x: pre_miRNA in x, chunks)[0] #수정 될
pre_miRNA를 가져옴
        num = chunks.index(beRevisedChunk) #수정 될 pre_miRNA의 위치 정보.
        for new_line in newDic[pre_miRNA]:
            wasChunk = chunks[num]
            chunks[num] = chunks[num].replace(new_line[0], new_line[1]) # 수정 전
read 이름을 수정 후 read 이름으로 바꿈
        for revised_chunk in chunks:
            writer.write('>' + revised_chunk)  #수정된 정보를 파일에 저장
    writer.close()

def parsing(mrdFile):
    mrdDic = dict()

def main(mrdFile):
    mrdDic = parsing(mrdFile)    #수정 전 정보를 가진 dictionary 생성
    newDic = comparing(mrdDic)   #수정된 정보를 가진 dictionary 생성
    revising(mrdFile, newDic)    #정보를 수정 후 파일에 저장

if __name__=='__main__':
    import sys
    import os
    mrdFile = sys.argv[1]  #mrd 파일
    main(mrdFile)
```

- 결과물

  - miRBase_new.mrd: 중복된 read들의 count가 보정된 파일

5) miRNA의 5′ heterogeneity와 mismatch을 고려한 발현량 분석

- 내용: miRDeep2의 결과를 이용하여, mature miRNA의 5′ heterogeneity (*onset/*offset)와 read의 3′ end의 mono- 또는 di-nucleotides mismatch (perfect/mismatch)를 고려하여 aligned reads를 4가지 그룹으로 나눠서 발현량을 재분석하는 in-house script를 이용함. 3′ end가 아닌 read 중간에 mismatch가 존재하는 모든 reads는 제거하는 과정을 거친 후, 발현량은 reads per million(RPM)으로 계산함. 3′ end의 mismatch가 존재하지 않을 때, annotation

상에 mature miRNA의 5′ end가 정확히 일치하는 것을 “onset_perfect”, 5′ end가 일치하지 않는 것을 “off_perfect”로 구분함. mismatch가 존재할 때, 5′ end의 일치 여부에 따라서 위와 동일하게 “onset_mismatch”, “offset_mismatch”로 align된 reads를 구분하여 그 세부 그룹별 정확한 발현량을 측정함.

*onset: read 의 5′ end 가 mature miRNA annotation 과 정확히 일치
*offset: read 의 5′end 가 mature miRNA annotation 과 불일치
*perfect: read 의 3′end 에 mismatch 가 없이 mature miRNA 에 맵핑
*mismatch: read 의 3′end 에 mono- 또는 di- nucleotides mismatch 가 존재함.

- 명령어의 예 (in-house script: mirdeepAnalysis.py, in-house script는 부록에 첨부되어 있음.)

```
python mirdeepAnalysis.py hsa hsa.hg19.gff3
~/mirDeepOutDir/miRBase_new.mrd ~/outputDir outputFile
```

-

input #1: miRNA species prefix를 입력 (ex) hsa, gga

- input #2: miRbase에서 제공하는 gff3 파일 형식의 annotation 파일을 입력
- input #3: miRDeep2의 quantifier를 이용해서 만들어진 결과물인 miRBase.mrd 파일을 입력
- input #4: output directory를 지정
- input #5: output file name을 지정

- 결과물

- _mirdeep2_expression.txt: miRDeep2 quantifier에서 만들어지는 동일한 형식으로 각 read가 어떤 세부 그룹에 속했는지를 나타내는 결과.

```
>hsa-mir-770
hsa-miR-770-5p  onset(perfect): 0      offset(perfect): 0
ffffffffffffffffff55555555555555555555555ffffffffffffffffffffffffffffffffffffffffffffffffffffff
..................uccaguaccacgugucagggcA......................................................
readCount: 1   misMatch:  1    onset_5p
>hsa-mir-6892
hsa-miR-6892-5p onset(perfect): 0      offset(perfect): 1
hsa-miR-6892-3p onset(perfect): 0      offset(perfect): 0
```

- _5p3p_perfectOnset_rate.txt: 5′ end가 annotation과 정확히 일치하고 mismatch가 존재하지 않는 mature miRNA의 발현량 (RPM)을 5p와 3p를 비교하여 fold change를 나타낸 결과.

```
Precursor_miRNA   Mature_5p_miRNA   RPM     Mature_3p_miRNA   RPM   Fold_change   Total_read_count
hsa-mir-3194      hsa-miR-3194-5p   0.00   hsa-miR-3194-3p     0.00    0.00    4183494
hsa-mir-675       hsa-miR-675-5p    0.72   hsa-miR-675-3p      7.89          0.09          4183494
```

- _mirdeep2_output.bed: mature miRNA를 각 그룹별로 bed format으로 작성함. 발현량의 상대적인 양은 발현량이 많을수록 진한 파란색의 color로 표기하였으며, offset은 onset의 경우보다 mature miRNA의 위치에 5′ end upstream -1, 3′ end downstream +1로 계산하여 position을 제공한 결과.

```
Chromosome    Start_position    End_position    Precursor_miRNA|Mature_miRNA|on/offset|match/mismatch
chr14         101318746         101318768        hsa-mir-770|hsa-miR-770-5p|onset|mismatch
chr7          143079777         143079801       hsa-mir-6892|hsa-miR-6892-5p|offset|perfect
```

- _OnsetOffset_Perfect_rate.txt: 각 mature miRNA의 5′ end가 정확히 일치하는 onset과 일치하지 않는 offset의 발현량 (RPM) 값을 fold change 값으로 제공한 결과.

```
Mature_miRNA  Onset_RPM     Offset_RPM    Fold_change   Total_read_count
hsa-miR-125b-5p      2193.62       15.30            143.39                  4183494
hsa-miR-1973         0.72          0.48                  1.50           4183494
```

- _OnsetPerfect_RPM.txt: 각 mature miRNA의 aligned read의 세부 그룹 중에 onset이고 mismatch가 없는 그룹의 발현량 (RPM)을 나타낸 결과.

```
Mature_miRNA  Onset_RPM     Read_count    Total_Read_count
hsa-miR-378c         385.80 1614          4183494
hsa-miR-495-3p       3.11          13            4183494
```

- _PerfectMismatch_onset_rate.txt: 각 mature miRNA에 onset이면서, perfect하게 align된 read와 mismatch를 가지고 있는 read의 RPM 값을 fold change로 나타낸 결과.

```
Mature_miRNA  Perfect_Onset_RPM   Mismatch_Onset_RPM  Fold_change   Total_read_count
hsa-miR-125b-5p      2193.62              2448.67             0.90          4183494
hsa-miR-495-3p       3.11                 0.24                13.00         4183494
```

- _tpEnd_mismatch.txt: 각 mature miRNA별로 3′end의 mono-와 di-nucleotide의 mismatch 수를 제공하는 결과.

```
>hsa-mir-125b-1
A:      9
C:      2
U:      6
G:      4
AA:     7
AC:     2
CA:     1
AU:     4
GA:     1
UA:     2
```

5. miRNA 차등발현 분석

1) 차등발현 분석을 위한 전처리 과정

- 내용: mirdeepAnalysis.py를 이용한 분석을 통해 얻은 결과 파일 (_OnsetPerfect_RPM.txt)을 이용해 차등 발현 분석을 위한 입력 파일을 만드는 과정을 진행함. 여러 개의 샘플이 존재할 때, 각 miRNA의 발현량을 샘플별로 정리하는 테이블을 출력함.
- 명령어의 예 (in-house script: integrateResult.py)

```
python integrateResult.py ~/inputDir/ 4 ~/outputDir/
```

-

input #1: input 파일들이 존재하는 directory 위치 (해당 input들은 sample1, sample2, sample3 같은 prefix로 되어 있어야 함.).

- input #2: input 파일의 개수
- input #3: output들이 작성될 directory 위치

```
import sys, os

inputDir = sys.argv[1]              ## input 파일들이 존재하는 directory 위치 (해당 input들은
sample1, sample2, sample3 같은 prefix로 되어있어야함)
inputFile_count = int(sys.argv[2])  ## input 파일의 개수
outputDir = sys.argv[3]             ## output들이 작성될 directory 위치

def read_onsetPerfect_file(inputDir, inputFile_count, filelist):
      ## 각 miRNA별로 RPM값이 정리된 table을 읽고 miRNA별로 multi sample의 값을 정리해주는 table을
만드는 함수
      mi_sampleReadDic = dict()
      for file in filelist:
            sampleNum = int(file.split("_")[1])
            fileOpen = open(inputDir + "/" + file, 'r')
            fileLines = fileOpen.readlines()
            for lines in fileLines:
                  matureMi = lines.strip().split("\t")[0]
                  RPM = float(lines.strip().split("\t")[1])
                  totalReads = lines.strip().split("\t")[-1]
                  if mi_sampleReadDic.has_key(matureMi):
                        mi_sampleReadDic[matureMi][sampleNum-1] = RPM
                  else:
                        mi_sampleReadDic[matureMi] = []
                        for i in range(0,inputFile_count):
mi_sampleReadDic[matureMi].append(0)
                        mi_sampleReadDic[matureMi][sampleNum-1] = RPM
      return mi_sampleReadDic

def nofilter_writeFile(matrixDic, outputDir, outputFileName):
      ## 전체 sample에서 miRNA의 발현량을 테이블로 정리함
      output = open(outputDir + "/" + outputFileName, 'w')
      for mi in matrixDic.keys():
            output.write(mi + "\t" + "\t".join(str(matrixDic[mi]).strip("[]").split(","))
+ "\n")
      output.close()

OnsetPerfect_RPM_list = filter(lambda x: x.endswith("OnsetPerfect_RPM.txt"),
os.listdir(inputDir))
OnsetPerfect_RPM_matrix = read_onsetPerfect_file(inputDir, inputFile_count,
OnsetPerfect_RPM_list)
nofilter_writeFile(OnsetPerfect_RPM_matrix, outputDir,
"sample_all_OnsetPerfect_RPM.txt")
```

- 결과물 (예시)

```
miRNA   sample1 sample2 sample3 sample4
miR-1   0.3     0.5     10      20
miR-2   1       3       3       1
miR-3   100     90      10      5
miR-4   50      50      1000    1000
miR-5   1000    900     300     400
miR-6   90      40      70      60
```

2) 차등발현 분석

- 내용: R 프로그램에서 edgeR 패키지를 이용해 miRNA들의 차등발현을 분석함. Unpaired 샘플들의 차등발현 분석을 위해 edgeR을 사용. 샘플들 사이에 존재하는 차이를 줄이기 위해 발현과 분산을 정규화하는 과정이 포함되어 있음.

- 명령어의 예 (edgeR 패키지)

```
data <- read.table("test.txt", header=T) # 예시 데이터.
data
#miRNA sample1 sample2 sample3 sample4
#miR-1   3e-01     0.5      10      20
#miR-2   1e+00     3.0       3       1
#miR-3   1e+02    90.0      10       5
#miR-4   5e+01    50.0    1000    1000
#miR-5   1e+03   900.0     300     400
#miR-6   9e+01    40.0      70      60
source('http://bioconductor.org/biocLite.R')       #bioconductor 에서 패키지를 불러
옴.

biocLite('edgeR')     #edgeR 설치.
library(edgeR)  #edgeR 패키지 로딩.
samples = c("sample1","sample2","sample3","sample4") #샘플들 이름.
groups = c("group1","group1","group2","group3") #샘플에 따른 그룹.
row.names(data) <- data$miRNA   #열 이름 지정.
data1 <- data[,samples]    #miRNA 이름을 제외한 expression 값을 지정.
dge <- DGEList(counts=data1, group=groups)      #리스트에 저장.
design <- model.matrix(~dge$samples$group)      #design 매트릭스 생성.
rownames(design) <- rownames(dge$samples) #design 매트릭스 열 이름 지정.
#분산에 대한 일반화 선형 모형을 생성 후 분산을 정규화.
dge_dispersion <- estimateGLMCommonDisp(dge,design)  #분산 추정
dge_tagwise <- estimateGLMTagwiseDisp(dge_dispersion,design) #tagwise 분산 추정
fit <- glmFit(dge_tagwise,design,dispersion=dge_tagwise$tagwitse.dispersion)
#일반화 선형 모형 생성
lrt <- glmLRT(fit)

#발현을 정규화
norm <- round(cpm(dge_tagwise,normalized.lib.sizes = T))
normExp <- merge(norm, topTags(lrt,n=10000), by.x="row.names", by.y="row.names")
names(normExp)[1]
normExp <- normExp[order(normExp$PValue),]
normExp
```

결과물

따른 p-value와 FDR값을 계산한 결과를 보여줌. P-value값의 순서대로 정렬되어 있음.

names	logFC	logCPM	LR	PValue	FDR
miR-4	5.445	19.159	103.996	2.026793e-24	1.216076e-23
miR-1	5.877	13.076	36.290	1.700158e-09	5.100473e-09
miR-3	-2.536	14.98	17.613	2.707493e-05	5.414985e-05
miR-6	1.038	15.774	3.047	8.086759e-02	1.213014e-01
miR-2	1.114	11.496	1.075	2.997250e-01	3.596700e-01
miR-5	-0.336	18.795	0.531	4.661002e-01	4.661002e-01

6. Reference

1) http://www.bioinformatics.babraham.ac.uk/projects/fastqc/
2) http://cutadapt.readthedocs.io/en/stable/index.html
3) https://www.perl.org/
4) https://www.mdc-berlin.de/8551903/en

7. 부록 (mirdeepAnalysis.py 소스코드)

```
import sys, os, re

species = sys.argv[1]         ## miRNA species symbol (ex) hsa, gga, mmu)
gff3FileName = sys.argv[2]          ## miRbase 에서 제공하는 miRNA annotation 파일 (gff3 포맷)
mirdeep2File = sys.argv[3]          ## mirdeep2 quantifier 를 통해 구한 alignment 결과 파일
outputDir = sys.argv[4]       ## 출력 파일이 쓰일 디렉토리
outputFileName = sys.argv[5]  ## 출력 파일의 파일명에 들어갈 접두사

class Locus:
      ## miRNA 의 위치를 class 로 구현
      __chrDict = dict()
      __senseDict = {'+':'+', '-':'-', '.':'.'}
      # chr = chromosome name (string)
      # sense = '+' or '-'
      # start,end = miRNA 의 start, end
      def __init__(self,chr,start,end,sense):
            coords = [start,end]
            coords.sort()
            if not(self.__chrDict.has_key(chr)): self.__chrDict[chr] = chr
            self._chr = self.__chrDict[chr]
            self._sense = self.__senseDict[sense]
            self._start = min(map(int, coords))
            self._end = max(map(int, coords))
      def chr(self): return self._chr
      def start(self): return self._start
      def end(self): return self._end
      def len(self): return self._end - self._start + 1
      def sense(self): return self._sense

def colorCode(exp,direction):
      ## miRNA 발현량 별로 color 를 구분하기 위한 함수
      cde=''
      if direction=='-':
            if exp>0 and exp<=1: cde =  '251,188,157'
            elif exp>1 and exp<5: cde ='248,149,112'
            elif exp>=5 and exp<10: cde = '244,108,66'
            elif exp>=10 and exp<50: cde = '240,50,0'
            elif exp>=50: cde = '150,28,0'
      else:
            if exp>0 and exp<=1: cde =  '166,178,239'
            elif exp>1 and exp<5: cde ='118,127,225'
            elif exp>=5 and exp<10: cde = '71,78,212'
            elif exp>=10 and exp<50: cde = '6,8,193'
            elif exp>=50: cde = '0,4,84'
      return cde

def getgff3(data):
      ## miRBase 에서 받은 annotation file 을 처리하는 함수
      ## miRNA_primary_transcript 와 miRNA 를 나눠서 locus 정보로 변환하여 dictionary 로 저장
      f = open(data, 'r')
      mature_annoDic = dict()
      pre_annoDic = dict()
      for line in f.readlines():
            if not line.startswith("#"):
                  raw_line = line.strip().split("\t")
                  if len(raw_line) > 1:
                        chr = raw_line[0]
                        miRNA_status = raw_line[2]
                        if miRNA_status == "miRNA_primary_transcript":
                              preName = raw_line[8].split(";")[2].split("=")[1]
                              strand = raw_line[6]
                              start = raw_line[3]
                              end = raw_line[4]
                              pre_miRNAlocus = Locus(chr, start, end, strand)
                              pre_annoDic[preName] = pre_miRNAlocus
                        if miRNA_status == "miRNA":
                              mName = raw_line[8].split(";")[2].split("=")[1]
                              miRNAname = preName + "|" + mName
                              strand = raw_line[6]
```

```
                              start = raw_line[3]
                              end = raw_line[4]
                              features = raw_line[8].split(";")
                              accessNum = features[1].split("=")[1]
                              miRNAlocus = Locus(chr, start, end, strand)
                              mature_annoDic[miRNAname] = miRNAlocus
      f.close()
      return pre_annoDic, mature_annoDic

class miRNAclass:
      ## pre-miRNA에 mapping된 read들을 세부 그룹으로 재정의하는 class
      ## 1) onset: annotation된 mature miRNA의 5'end 위치와 정확하게 일치하는 read 그룹
      ## 2) offset: annotation된 mature miRNA의 5'end 위치와 일치하지 않는 read 그룹
      ## 3) perfect: read에 mismatch가 존재하지 않은채 mapping된 read 그룹
      ## 4) mismatch: read에 mismatch가 존재하는 read 그룹
      ## 5) M: miR-5p와 miR-3p가 annotation되어 있지 않은 miRNA는 M으로 인덱싱이 되어 있음. 이를 5p/3p 위치에 따라 나눔
      ## 6) miRNA annotation된 위치로 upstream 2nt, downstream 5nt 위치 밖의 read들은 undefined 그룹으로 분류
      def __init__(self, exp, seqList):
            self.__exp = exp
            self.__seqList = seqList

            def define_on_off(self, seq):
                  p = re.compile("[a-zA-Z]")
                  ntIndex = p.search(seq).start()
                  if (ntIndex < len(seq)/2):
                        if not (self.__exp.find("5") ==-1):
                              if not (self.__exp[ntIndex:ntIndex+len(seq.strip("."))].find("5") ==
-1):
                                    if ntIndex == self.__exp.find("5"): self._on_off_class =
"onset_5p"
                                    elif ntIndex != self.__exp.find("5"):
                                          if (self.__exp.find("5") -2 <= ntIndex and ntIndex <=
self.__exp.find("5") +5): self._on_off_class = "offset_5p"
                                          else: self._on_off_class = "undefined"
                              else:
                                    if self.__exp.find("3") ==-1: self._on_off_class = "star"
                                    else: self._on_off_class = "undefined"
                        elif not(self.__exp.find("M") ==-1):
                              if not (self.__exp[ntIndex:ntIndex+len(seq.strip("."))].find("M") ==
-1):
                                    if ntIndex == self.__exp.find("M"): self._on_off_class =
"onset_M_5p"
                                    elif ntIndex != self.__exp.find("M"):
                                          if (self.__exp.find("M") -2 <= ntIndex and ntIndex <=
self.__exp.find("M") +5): self._on_off_class = "offset_M_5p"
                                          else: self._on_off_class = "undefined"
                              else: self._on_off_class = "star"
                        else: self._on_off_class = "star"
                  elif (ntIndex >= len(seq)/2):
                        if not (self.__exp.find("3") ==-1):
                              if not (self.__exp[ntIndex:ntIndex+len(seq.strip("."))].find("3") ==
-1):
                                    if ntIndex == self.__exp.find("3"): self._on_off_class =
"onset_3p"
                                    elif ntIndex != self.__exp.find("3"):
                                          if (self.__exp.find("3") -2 <= ntIndex and ntIndex <=
self.__exp.find("3") +5): self._on_off_class = "offset_3p"
                                          else: self._on_off_class = "undefined"
                              else:
                                    if self.__exp.find("5") ==-1: self._on_off_class = "star"
                                    else: self._on_off_class = "undefined"
                        elif not(self.__exp.find("M") ==-1):
                              if not (self.__exp[ntIndex:ntIndex+len(seq.strip("."))].find("M") ==
-1):
                                    if ntIndex ==
self.__exp[len(self.__exp)/2:].find("M")+len(self.__exp)/2: self._on_off_class = "onset_M_3p"
                                    elif ntIndex !=
self.__exp[len(self.__exp)/2:].find("M")+len(self.__exp)/2:
                                          if
(self.  exp[len(self.  exp)/2:].find("M")+len(self.  exp)/2 -2 <= ntIndex and ntIndex
```

```
<=self.__exp[len(self.__exp)/2:].find("M")+len(self.__exp)/2 +5): self._on_off_class =
"offset_M_3p"
                                        else: self._on_off_class = "undefined"
                              else: self._on_off_class = "star"
                        else: self._on_off_class = "star"
                  return self._on_off_class

            def star_on_off(self, star_dict):
                  star_onset_readCount = []
                  star_offset_readCount = []
                  star_onset_seq =
star_dict.keys()[star_dict.values().index(max(star_dict.values()))]
                  star_onset_readCount.append(star_dict[star_onset_seq])
                  p = re.compile("[a-zA-Z]")
                  star_onset_ntIndex = p.search(star_onset_seq).start()
                  del star_dict[star_onset_seq]
                  for star_seq in star_dict.keys():
                        if star_onset_ntIndex == p.search(star_seq).start():
star_onset_readCount.append(star_dict[star_seq])
                        elif star_onset_ntIndex-2 < p.search(star_seq).start() and
p.search(star_seq).start() < star_onset_ntIndex +5:
star_offset_readCount.append(star_dict[star_seq])
                  self._star_onset_ntIndex = star_onset_ntIndex
                  self._star_onset_length = len(star_onset_seq.strip("."))
                  self._star_onset_count = sum(star_onset_readCount)
                  self._star_offset_count = sum(star_offset_readCount)
                  return self._star_onset_ntIndex, self._star_onset_length, self._star_onset_count,
self._star_offset_count

            perfect_onset_5p = []
            perfect_offset_5p = []
            perfect_onset_3p = []
            perfect_offset_3p = []
            perfect_onset_M_5p = []
            perfect_offset_M_5p = []
            perfect_onset_M_3p = []
            perfect_offset_M_3p = []
            perfect_undefined = []
            perfect_star = dict()
            mismatch_onset_5p = []
            mismatch_offset_5p = []
            mismatch_onset_3p = []
            mismatch_offset_3p = []
            mismatch_onset_M_5p = []
            mismatch_offset_M_5p = []
            mismatch_onset_M_3p = []
            mismatch_offset_M_3p = []
            mismatch_undefined = []
            mismatch_star = dict()
            self._mono_mismatch = dict()
            self._di_mismatch = dict()
            self._sequenceList = []
            for seqLineDict in self.__seqList:
                  readCount = int(seqLineDict.keys()[0].split("_")[2][1:])
                  misMatchScore = int(seqLineDict.keys()[0].split("_")[3][2:])
                  seq = seqLineDict.values()[0]
                  if misMatchScore == 0:
                        on_off_class = define_on_off(self, seq)
                        if on_off_class == "onset_5p": perfect_onset_5p.append(readCount)
                        if on_off_class == "offset_5p": perfect_offset_5p.append(readCount)
                        if on_off_class == "onset_3p": perfect_onset_3p.append(readCount)
                        if on_off_class == "offset_3p": perfect_offset_3p.append(readCount)
                        if on_off_class == "onset_M_5p": perfect_onset_M_5p.append(readCount)
                        if on_off_class == "offset_M_5p": perfect_offset_M_5p.append(readCount)
                        if on_off_class == "onset_M_3p": perfect_onset_M_3p.append(readCount)
                        if on_off_class == "offset_M_3p": perfect_offset_M_3p.append(readCount)
                        if on_off_class == "undefined": perfect_undefined.append(readCount)
                        if on_off_class == "star": perfect_star[seq] = readCount
                        self._sequenceList.append(seq +'\t'+ "readCount: "+ `readCount` +'\t' +
"misMatch: " + `misMatchScore`+"\t"+ on_off_class)
                  else:
                        nt = seq.strip(".")
                        upper = re.compile("[ACGU]")
                        under = re.compile("[acgu]")
                        if len(map(lambda x: x, upper.finditer(nt[:len(nt)-2]))) == 0:
```

```
                              if not (misMatchScore == 1 and upper.search(nt[len(nt)-2]) != None):
                                    if under.search(nt[len(nt)-2:]):
                                          if self._mono_mismatch.has_key(nt[-1]):
self._mono_mismatch[nt[-1]] += 1
                                          else: self._mono_mismatch[nt[-1]] = 0;
self._mono_mismatch[nt[-1]] += 1
                                    elif under.search(nt[len(nt)-2:]) == None:
                                          if self._di_mismatch.has_key(nt[len(nt)-2:]):
self._di_mismatch[nt[len(nt)-2:]] += 1
                                          else: self._di_mismatch[nt[len(nt)-2:]] = 0;
self._di_mismatch[nt[len(nt)-2:]] += 1
                                    on_off_class = define_on_off(self, seq)
                                    if on_off_class == "onset_5p":
mismatch_onset_5p.append(readCount)
                                    if on_off_class == "offset_5p":
mismatch_offset_5p.append(readCount)
                                    if on_off_class == "onset_3p":
mismatch_onset_3p.append(readCount)
                                    if on_off_class == "offset_3p":
mismatch_offset_3p.append(readCount)
                                    if on_off_class == "onset_M_5p":
mismatch_onset_M_5p.append(readCount)
                                    if on_off_class == "offset_M_5p":
mismatch_offset_M_5p.append(readCount)
                                    if on_off_class == "onset_M_3p":
mismatch_onset_M_3p.append(readCount)
                                    if on_off_class == "offset_M_3p":
mismatch_offset_M_3p.append(readCount)
                                    if on_off_class == "undefined":
mismatch_undefined.append(readCount)
                                    if on_off_class == "star": mismatch_star[seq] = readCount
                                    self._sequenceList.append(seq +"\t"+ "readCount: "+
`readCount`+ '\t' + "misMatch: " + `misMatchScore`+"\t"+ on_off_class)
            self._perfect_star_onset_ntIndex = 0
            self._perfect_star_onset_length = 0
            self._perfect_star_onset_count = 0
            self._perfect_star_offset_count = 0
            self._mismatch_star_onset_ntIndex = 0
            self._mismatch_star_onset_length = 0
            self._mismatch_star_onset_count = 0
            self._mismatch_star_offset_count = 0
            if not(len(perfect_star.keys()) == 0):
                  perfect_star_onset_ntIndex, perfect_star_onset_length, perfect_star_onset_count,
perfect_star_offset_count = star_on_off(self, perfect_star)
                  self._perfect_star_onset_ntIndex = perfect_star_onset_ntIndex
                  self._perfect_star_onset_length = perfect_star_onset_length
                  self._perfect_star_onset_count = perfect_star_onset_count
                  self._perfect_star_offset_count = perfect_star_offset_count
            if not(len(mismatch_star.keys()) == 0):
                  mismatch_star_onset_ntIndex, mismatch_star_onset_length,
mismatch_star_onset_count, mismatch_star_offset_count = star_on_off(self, mismatch_star)
                  self._mismatch_star_onset_ntIndex = mismatch_star_onset_ntIndex
                  self._mismatch_star_onset_length = mismatch_star_onset_length
                  self._mismatch_star_onset_count = mismatch_star_onset_count
                  self._mismatch_star_offset_count = mismatch_star_offset_count
            self._perfect_onset_5p_count = sum(perfect_onset_5p)
            self._perfect_offset_5p_count = sum(perfect_offset_5p)
            self._perfect_onset_3p_count = sum(perfect_onset_3p)
            self._perfect_offset_3p_count = sum(perfect_offset_3p)
            self._perfect_onset_M_5p_count = sum(perfect_onset_M_5p)
            self._perfect_offset_M_5p_count = sum(perfect_offset_M_5p)
            self._perfect_onset_M_3p_count = sum(perfect_onset_M_3p)
            self._perfect_offset_M_3p_count = sum(perfect_offset_M_3p)
            self._perfect_undefined_count = sum(perfect_undefined)
            self._mismatch_onset_5p_count = sum(mismatch_onset_5p)
            self._mismatch_offset_5p_count = sum(mismatch_offset_5p)
            self._mismatch_onset_3p_count = sum(mismatch_onset_3p)
            self._mismatch_offset_3p_count = sum(mismatch_offset_3p)
            self._mismatch_onset_M_5p_count = sum(mismatch_onset_M_5p)
            self._mismatch_offset_M_5p_count = sum(mismatch_offset_M_5p)
            self._mismatch_onset_M_3p_count = sum(mismatch_onset_M_3p)
            self._mismatch_offset_M_3p_count = sum(mismatch_offset_M_3p)
            self._mismatch_undefined_count = sum(mismatch_undefined)

      def exp(self): return self.__exp
      def seqList(self): return self.  seqList
```

```
      def perfect_onset_5p_count(self): return self._perfect_onset_5p_count
      def perfect_offset_5p_count(self): return self._perfect_offset_5p_count
      def perfect_onset_3p_count(self): return self._perfect_onset_3p_count
      def perfect_offset_3p_count(self): return self._perfect_offset_3p_count
      def perfect_onset_M_5p_count(self): return self._perfect_onset_M_5p_count
      def perfect_offset_M_5p_count(self): return self._perfect_offset_M_5p_count
      def perfect_onset_M_3p_count(self): return self._perfect_onset_M_3p_count
      def perfect_offset_M_3p_count(self): return self._perfect_offset_M_3p_count
      def perfect_undefined_count(self): return self._perfect_undefined_count

      def mismatch_onset_5p_count(self): return self._mismatch_onset_5p_count
      def mismatch_offset_5p_count(self): return self._mismatch_offset_5p_count
      def mismatch_onset_3p_count(self): return self._mismatch_onset_3p_count
      def mismatch_offset_3p_count(self): return self._mismatch_offset_3p_count
      def mismatch_onset_M_5p_count(self): return self._mismatch_onset_M_5p_count
      def mismatch_offset_M_5p_count(self): return self._mismatch_offset_M_5p_count
      def mismatch_onset_M_3p_count(self): return self._mismatch_onset_M_3p_count
      def mismatch_offset_M_3p_count(self): return self._mismatch_offset_M_3p_count
      def mismatch_undefined_count(self): return self._mismatch_undefined_count
      def classifiedSequenceList(self): return self._sequenceList

      def perfect_star_onset_ntIndex(self): return self._perfect_star_onset_ntIndex
      def perfect_star_onset_length(self): return self._perfect_star_onset_length
      def perfect_star_onset_count(self): return self._perfect_star_onset_count
      def perfect_star_offset_count(self): return self._perfect_star_offset_count

      def mismatch_star_onset_ntIndex(self): return self._mismatch_star_onset_ntIndex
      def mismatch_star_onset_length(self): return self._mismatch_star_onset_length
      def mismatch_star_onset_count(self): return self._mismatch_star_onset_count
      def mismatch_star_offset_count(self): return self._mismatch_star_offset_count

      def mismatch_mono(self): return     self._mono_mismatch
      def mismatch_di(self): return self._di_mismatch


def write_bedFile(pre_annoDic, mature_annoDic, mirdeep2_dict, outputDir, outputFileName,
totalReadCount):
      ## 정리된 miRNA의 그룹별 read 수를 bed 파일 형식으로 출력해주는 함수
      def makeOnsetBedLine(lineName, readCount, locusObj):
            line = locusObj.chr() + "\t" + `locusObj.start()` + "\t" + `locusObj.end()` + "\t" +
lineName + "\t" + `readCount` + "\t" + locusObj.sense() + "\t" + `locusObj.start()` + "\t" +
`locusObj.end()` + "\t" + UM.colorCode(readCount, locusObj.sense())
            return line

      def makeOffsetBedLine(lineName, readCount, locusObj):
            line = locusObj.chr() + "\t" + `locusObj.start() -2` + "\t" + `locusObj.end()+2` + "\t"
+ lineName + "\t" + `readCount` + "\t" + locusObj.sense() + "\t" + `locusObj.start()-2` + "\t" +
`locusObj.end()+2` + "\t" + UM.colorCode(readCount, locusObj.sense())
            return line

      def checkStar(pre_annoDic, miRID, miRclass):
            starLine = []
            preID = miRID.split("|")[0]
            def makeOnsetStarLine(pre_annoDic, preID, miRclass, bedlineName):
                  if pre_annoDic.has_key(preID):
                        if pre_annoDic[preID].sense() == "+":
                              starStart =
str(pre_annoDic[preID].start()+miRclass.perfect_star_onset_ntIndex())
                              starEnd =
str(pre_annoDic[preID].start()+miRclass.perfect_star_onset_ntIndex() +
miRclass.perfect_star_onset_length())
                              line = pre_annoDic[preID].chr() + "\t" + starStart + "\t" + starEnd +
"\t" + bedlineName + "\t" + `miRclass.perfect_star_onset_count()` + "\t" +
pre_annoDic[preID].sense() + "\t" + starStart + "\t" + starEnd + "\t" +
UM.colorCode(miRclass.perfect_star_onset_count(), pre_annoDic[preID].sense())
                        elif pre_annoDic[preID].sense() == "-":
                              starStart = str(pre_annoDic[preID].end()-
miRclass.perfect_star_onset_ntIndex())
                              starEnd = str(pre_annoDic[preID].end()-
miRclass.perfect_star_onset_ntIndex() - miRclass.perfect_star_onset_length())
                              line = pre_annoDic[preID].chr() + "\t" + starEnd + "\t" + starStart +
"\t" + bedlineName + "\t" + `miRclass.perfect_star_onset_count()` + "\t" +
pre_annoDic[preID].sense() + "\t" + starStart + "\t" + starEnd + "\t" +
UM.colorCode(miRclass.perfect_star_onset_count(), pre_annoDic[preID].sense())
                        return line
```

```
            def makeOffsetStarLine(pre_annoDic, preID, miRclass, bedlineName):
                  if pre_annoDic[preID].sense() == "+":
                        starStart =
str(pre_annoDic[preID].start()+miRclass.perfect_star_onset_ntIndex()-2)
                        starEnd =
str(pre_annoDic[preID].start()+miRclass.perfect_star_onset_ntIndex() +
miRclass.perfect_star_onset_length()+2)
                        line = pre_annoDic[preID].chr() + "\t" + starStart + "\t" + starEnd + "\t"
+ bedlineName + "\t" + `miRclass.perfect_star_onset_count()` + "\t" + pre_annoDic[preID].sense() +
"\t" + starStart + "\t" + starEnd + "\t" + UM.colorCode(miRclass.perfect_star_onset_count(),
pre_annoDic[preID].sense())
                  elif pre_annoDic[preID].sense() == "-":
                        starStart = str(pre_annoDic[preID].end()-
miRclass.perfect_star_onset_ntIndex()+2)
                        starEnd = str(pre_annoDic[preID].end()-
miRclass.perfect_star_onset_ntIndex() - miRclass.perfect_star_onset_length()-2)
                        line = pre_annoDic[preID].chr() + "\t" + starEnd + "\t" + starStart + "\t"
+ bedlineName + "\t" + `miRclass.perfect_star_onset_count()` + "\t" + pre_annoDic[preID].sense() +
"\t" + starStart + "\t" + starEnd + "\t" + UM.colorCode(miRclass.perfect_star_onset_count(),
pre_annoDic[preID].sense())
                  return line
            if pre_annoDic.has_key(preID):
                  ## star perfect match
                  if not miRclass.perfect_star_onset_count() == 0:
                        if miRID.split("-")[-1] in ["5p", "3p"]:
                              bedlineName = "-".join(miRID.split("-")[:-1])+"-star|onset|perfect"
                              line = makeOnsetStarLine(pre_annoDic, preID, miRclass, bedlineName)
                              starLine.append(line)
                        else:
                              bedlineName = miRID + "-star|onset|perfect"
                              line = makeOnsetStarLine(pre_annoDic, preID, miRclass, bedlineName)
                              starLine.append(line)

                  if not miRclass.perfect_star_offset_count() == 0:
                        if miRID.split("-")[-1] in ["5p", "3p"]:
                              bedlineName = "-".join(miRID.split("-")[:-1])+"-star|offset|perfect"
                              line = makeOffsetStarLine(pre_annoDic, preID, miRclass, bedlineName)
                              starLine.append(line)
                        else:
                              bedlineName = miRID + "-star|offset|perfect"
                              line = makeOffsetStarLine(pre_annoDic, preID, miRclass, bedlineName)
                              starLine.append(line)
                  ## star mismatch
                  if not miRclass.mismatch_star_onset_count() == 0:
                        if miRID.split("-")[-1] in ["5p", "3p"]:
                              bedlineName = "-".join(miRID.split("-")[:-1])+"-star|onset|mismatch"
                              line = makeOnsetStarLine(pre_annoDic, preID, miRclass, bedlineName)
                              starLine.append(line)
                        else:
                              bedlineName = miRID + "-star|onset|mismatch"
                              line = makeOnsetStarLine(pre_annoDic, preID, miRclass, bedlineName)
                              starLine.append(line)

                  if not miRclass.mismatch_star_offset_count() == 0:
                        if miRID.split("-")[-1] in ["5p", "3p"]:
                              bedlineName = "-".join(miRID.split("-")[:-1])+"-star|offset|mismatch"
                              line = makeOffsetStarLine(pre_annoDic, preID, miRclass, bedlineName)
                              starLine.append(line)
                        else:
                              bedlineName = miRID + "-star|offset|mismatch"
                              line = makeOffsetStarLine(pre_annoDic, preID, miRclass, bedlineName)
                              starLine.append(line)
            return starLine

      bedLineList = []
      for keys in mirdeep2_dict.keys():
            miRclass = mirdeep2_dict[keys]["miRclass"]
            if not len(mirdeep2_dict[keys]["mir5pID"]) == 0:
                  mir5p = keys + "|" + mirdeep2_dict[keys]["mir5pID"]
                  star5pLineList = checkStar(pre_annoDic, mir5p, miRclass)
                  if not star5pLineList == 0:  bedLineList = bedLineList+star5pLineList
                  if mature_annoDic.has_key(mir5p):
                        if not miRclass.perfect_onset_5p_count() == 0:
                              bedlineName = mir5p + "|onset|perfect"
                              bedLineList.append(makeOnsetBedLine(bedlineName,
miRclass.perfect_onset_5p_count(), mature_annoDic[mir5p]))
```

```
                        if not miRclass.perfect_offset_5p_count() == 0:
                              bedlineName = mir5p + "|offset|perfect"
                              bedLineList.append(makeOffsetBedLine(bedlineName,
miRclass.perfect_offset_5p_count(), mature_annoDic[mir5p]))
                        if not miRclass.mismatch_onset_5p_count() == 0:
                              bedlineName = mir5p + "|onset|mismatch"
                              bedLineList.append(makeOnsetBedLine(bedlineName,
miRclass.mismatch_onset_5p_count(), mature_annoDic[mir5p]))
                        if not miRclass.mismatch_offset_5p_count() == 0:
                              bedlineName = mir5p + "|offset|mismatch"
                              bedLineList.append(makeOffsetBedLine(bedlineName,
miRclass.mismatch_offset_5p_count(), mature_annoDic[mir5p]))

            if not len(mirdeep2_dict[keys]["mir3pID"]) == 0:
                  mir3p = keys + "|" + mirdeep2_dict[keys]["mir3pID"]
                  star3pLineList = checkStar(pre_annoDic, mir3p, miRclass)
                  if not star3pLineList == 0: bedLineList = bedLineList+star3pLineList
                  if mature_annoDic.has_key(mir3p):
                        if not miRclass.perfect_onset_3p_count() == 0:
                              bedlineName = mir3p + "|onset|perfect"
                              bedLineList.append(makeOnsetBedLine(bedlineName,
miRclass.perfect_onset_3p_count(), mature_annoDic[mir3p]))
                        if not miRclass.perfect_offset_3p_count() == 0:
                              bedlineName = mir3p + "|offset|perfect"
                              bedLineList.append(makeOffsetBedLine(bedlineName,
miRclass.perfect_offset_3p_count(), mature_annoDic[mir3p]))
                        if not miRclass.mismatch_onset_3p_count() == 0:
                              bedlineName = mir3p + "|onset|mismatch"
                              bedLineList.append(makeOnsetBedLine(bedlineName,
miRclass.mismatch_onset_3p_count(), mature_annoDic[mir3p]))
                        if not miRclass.mismatch_offset_3p_count() == 0:
                              bedlineName = mir3p + "|offset|mismatch"
                              bedLineList.append(makeOffsetBedLine(bedlineName,
miRclass.mismatch_offset_3p_count(), mature_annoDic[mir3p]))

            if not len(mirdeep2_dict[keys]["mirID"]) == 0:
                  mirM = keys + "|" + mirdeep2_dict[keys]["mirID"]
                  starMLineList = checkStar(pre_annoDic, mirM, miRclass)
                  if not starMLineList == 0: bedLineList = bedLineList+starMLineList
                  if mature_annoDic.has_key(mirM):
                        if not miRclass.perfect_onset_M_5p_count() == 0:
                              bedlineName = mirM + "|onset|perfect"
                              bedLineList.append(makeOnsetBedLine(bedlineName,
miRclass.perfect_onset_M_5p_count(), mature_annoDic[mirM]))
                        if not miRclass.perfect_offset_M_5p_count() == 0:
                              bedlineName = mirM + "|offset|perfect"
                              bedLineList.append(makeOffsetBedLine(bedlineName,
miRclass.perfect_offset_M_5p_count(), mature_annoDic[mirM]))
                        if not miRclass.mismatch_onset_M_5p_count() == 0:
                              bedlineName = mirM + "|onset|mismatch"
                              bedLineList.append(makeOnsetBedLine(bedlineName,
miRclass.mismatch_onset_M_5p_count(), mature_annoDic[mirM]))
                        if not miRclass.mismatch_offset_M_5p_count() == 0:
                              bedlineName = mirM + "|offset|mismatch"
                              bedLineList.append(makeOffsetBedLine(bedlineName,
miRclass.mismatch_offset_M_5p_count(), mature_annoDic[mirM]))
                        if not miRclass.perfect_onset_M_3p_count() == 0:
                              bedlineName = mirM + "|onset|perfect"
                              bedLineList.append(makeOnsetBedLine(bedlineName,
miRclass.perfect_onset_M_3p_count(), mature_annoDic[mirM]))
                        if not miRclass.perfect_offset_M_3p_count() == 0:
                              bedlineName = mirM + "|offset|perfect"
                              bedLineList.append(makeOffsetBedLine(bedlineName,
miRclass.perfect_offset_M_3p_count(), mature_annoDic[mirM]))
                        if not miRclass.mismatch_onset_M_3p_count() == 0:
                              bedlineName = mirM + "|onset|mismatch"
                              bedLineList.append(makeOnsetBedLine(bedlineName,
miRclass.mismatch_onset_M_3p_count(), mature_annoDic[mirM]))
                        if not miRclass.mismatch_offset_M_3p_count() == 0:
                              bedlineName = mirM + "|offset|mismatch"
                              bedLineList.append(makeOffsetBedLine(bedlineName,
miRclass.mismatch_offset_M_3p_count(), mature_annoDic[mirM]))

            bedFile = open(outputDir + "/" + outputFileName + "_mirdeep2_output.bed", 'w')
            bedFile.write("\n".join(bedLineList))
            bedFile.close()
```

```
def write_classFile(mirdeep2_dict, preMi_seqDic, outputDir, outputFileName):
      ## pre-miRNA에 mapping된 read를 보여주는 데이터로, 각 read별로 read count와 mismatch 수와 어떤 그룹으로 분류되어있는지를 종합적으로 확인할 수 있는 파일
      expressionFile = open(outputDir + "/" +outputFileName + "_mirdeep2_expression.txt", 'w')
      for preMi in mirdeep2_dict.keys():
            miRclass =mirdeep2_dict[preMi]["miRclass"]
            if not len(preMi_seqDic[preMi]) ==0:
                  expressionFile.write(">" + preMi+"\n")
                  if not len(mirdeep2_dict[preMi]["mir5pID"]) == 0:
expressionFile.write(mirdeep2_dict[preMi]["mir5pID"]+ "\t" + "onset(perfect): " +
`miRclass.perfect_onset_5p_count()` + "\t"+ "offset(perfect): " +
`miRclass.perfect_offset_5p_count()` + "\n")
                  if not len(mirdeep2_dict[preMi]["mir3pID"]) == 0:
expressionFile.write(mirdeep2_dict[preMi]["mir3pID"]+ "\t" + "onset(perfect): " +
`miRclass.perfect_onset_3p_count()` + "\t"+ "offset(perfect): " +
`miRclass.perfect_offset_3p_count()` + "\n")
                  if not len(mirdeep2_dict[preMi]["mirID"]) == 0:
expressionFile.write(mirdeep2_dict[preMi]["mirID"]+ "\t" + "onset(perfect): " +
`miRclass.perfect_onset_M_5p_count() + miRclass.perfect_onset_M_3p_count()` + "\t"+
"offset(perfect): " +  `miRclass.perfect_offset_M_5p_count() +
miRclass.perfect_offset_M_3p_count()` + "\n")
                  if not len(mirdeep2_dict[preMi]["exp"]) == 0:
expressionFile.write(mirdeep2_dict[preMi]["exp"]+"\n")
                  expressionFile.write('\n'.join(preMi_seqDic[preMi]))
                  expressionFile.write('\n')
      expressionFile.close()


def write_statisticFile(mirdeep2_dict, preMi_seqDic, outputDir, outputFileName, totalReadCount):
      ## miRNA 그룹별로 발현량 (rpm)과 fold change를 정리해주는 출력파일을 만드는 함수
      perfectOnset_RPMdict = dict()
      mismatchOnset_RPMdict = dict()
      perfectOffset_RPMdict = dict()

      perfectOnset_ReadDict = dict()
      mismatchOnset_ReadDict = dict()
      perfectOffset_ReadDict = dict()

      for preMi in mirdeep2_dict.keys():
            miRclass = mirdeep2_dict[preMi]["miRclass"]
            if not len(preMi_seqDic[preMi]) == 0:
                  if not len(mirdeep2_dict[preMi]["mir5pID"]) == 0:
                        if perfectOnset_RPMdict.has_key(mirdeep2_dict[preMi]["mir5pID"]):
                              perfectOnset_RPMdict[mirdeep2_dict[preMi]["mir5pID"]] +=
(miRclass.perfect_onset_5p_count() / float(totalReadCount) * 1000000)
                        if not (perfectOnset_RPMdict.has_key(mirdeep2_dict[preMi]["mir5pID"])):
                              perfectOnset_RPMdict[mirdeep2_dict[preMi]["mir5pID"]] = 0
                              perfectOnset_RPMdict[mirdeep2_dict[preMi]["mir5pID"]] +=
(miRclass.perfect_onset_5p_count() / float(totalReadCount) * 1000000)

                        if perfectOnset_ReadDict.has_key(mirdeep2_dict[preMi]["mir5pID"]):
                              perfectOnset_ReadDict[mirdeep2_dict[preMi]["mir5pID"]] +=
miRclass.perfect_onset_5p_count()
                        if not (perfectOnset_ReadDict.has_key(mirdeep2_dict[preMi]["mir5pID"])):
                              perfectOnset_ReadDict[mirdeep2_dict[preMi]["mir5pID"]] = 0
                              perfectOnset_ReadDict[mirdeep2_dict[preMi]["mir5pID"]] +=
miRclass.perfect_onset_5p_count()

                  if not len(mirdeep2_dict[preMi]["mir3pID"]) == 0:
                        if perfectOnset_RPMdict.has_key(mirdeep2_dict[preMi]["mir3pID"]):
                              perfectOnset_RPMdict[mirdeep2_dict[preMi]["mir3pID"]] +=
(miRclass.perfect_onset_3p_count() / float(totalReadCount) * 1000000)
                        if not(perfectOnset_RPMdict.has_key(mirdeep2_dict[preMi]["mir3pID"])):
                              perfectOnset_RPMdict[mirdeep2_dict[preMi]["mir3pID"]] = 0
                              perfectOnset_RPMdict[mirdeep2_dict[preMi]["mir3pID"]] +=
(miRclass.perfect_onset_3p_count() / float(totalReadCount) * 1000000)

                        if perfectOnset_ReadDict.has_key(mirdeep2_dict[preMi]["mir3pID"]):
                              perfectOnset_ReadDict[mirdeep2_dict[preMi]["mir3pID"]] +=
miRclass.perfect_onset_3p_count()
                        if not(perfectOnset_ReadDict.has_key(mirdeep2_dict[preMi]["mir3pID"])):
                              perfectOnset_ReadDict[mirdeep2_dict[preMi]["mir3pID"]] = 0
                              perfectOnset_ReadDict[mirdeep2_dict[preMi]["mir3pID"]] +=
miRclass.perfect_onset_3p_count()
```

```
                if not len(mirdeep2_dict[preMi]["mirID"]) == 0:
                      if perfectOnset_RPMdict.has_key(mirdeep2_dict[preMi]["mirID"]):
                            perfectOnset_RPMdict[mirdeep2_dict[preMi]["mirID"]] +=
((miRclass.perfect_onset_M_5p_count() +miRclass.perfect_onset_M_3p_count()) / float(totalReadCount)
* 1000000)
                      if not (perfectOnset_RPMdict.has_key(mirdeep2_dict[preMi]["mirID"])):
                            perfectOnset_RPMdict[mirdeep2_dict[preMi]["mirID"]] = 0
                            perfectOnset_RPMdict[mirdeep2_dict[preMi]["mirID"]] +=
((miRclass.perfect_onset_M_5p_count() +miRclass.perfect_onset_M_3p_count()) / float(totalReadCount)
* 1000000)

                      if perfectOnset_ReadDict.has_key(mirdeep2_dict[preMi]["mirID"]):
                            perfectOnset_ReadDict[mirdeep2_dict[preMi]["mirID"]] +=
(miRclass.perfect_onset_M_5p_count() +miRclass.perfect_onset_M_3p_count())
                      if not (perfectOnset_ReadDict.has_key(mirdeep2_dict[preMi]["mirID"])):
                            perfectOnset_ReadDict[mirdeep2_dict[preMi]["mirID"]] = 0
                            perfectOnset_ReadDict[mirdeep2_dict[preMi]["mirID"]] +=
(miRclass.perfect_onset_M_5p_count() +miRclass.perfect_onset_M_3p_count())

                if not len(mirdeep2_dict[preMi]["mir5pID"]) == 0:
                      if mismatchOnset_RPMdict.has_key(mirdeep2_dict[preMi]["mir5pID"]):
mismatchOnset_RPMdict[mirdeep2_dict[preMi]["mir5pID"]] += (miRclass.mismatch_onset_5p_count() /
float(totalReadCount) * 1000000)
                      else: mismatchOnset_RPMdict[mirdeep2_dict[preMi]["mir5pID"]] = 0;
mismatchOnset_RPMdict[mirdeep2_dict[preMi]["mir5pID"]] += (miRclass.mismatch_onset_5p_count() /
float(totalReadCount) * 1000000)
                if not len(mirdeep2_dict[preMi]["mir3pID"]) == 0:
                      if mismatchOnset_RPMdict.has_key(mirdeep2_dict[preMi]["mir3pID"]):
mismatchOnset_RPMdict[mirdeep2_dict[preMi]["mir3pID"]] += (miRclass.mismatch_onset_3p_count() /
float(totalReadCount) * 1000000)
                      else: mismatchOnset_RPMdict[mirdeep2_dict[preMi]["mir3pID"]] = 0;
mismatchOnset_RPMdict[mirdeep2_dict[preMi]["mir3pID"]] += (miRclass.mismatch_onset_3p_count() /
float(totalReadCount) * 1000000)
                if not len(mirdeep2_dict[preMi]["mirID"]) == 0:
                      if mismatchOnset_RPMdict.has_key(mirdeep2_dict[preMi]["mirID"]):
mismatchOnset_RPMdict[mirdeep2_dict[preMi]["mirID"]] += ((miRclass.mismatch_onset_M_5p_count()
+miRclass.mismatch_onset_M_3p_count()) / float(totalReadCount) * 1000000)
                      else: mismatchOnset_RPMdict[mirdeep2_dict[preMi]["mirID"]] = 0;
mismatchOnset_RPMdict[mirdeep2_dict[preMi]["mirID"]] += ((miRclass.mismatch_onset_M_5p_count()
+miRclass.mismatch_onset_M_3p_count()) / float(totalReadCount) * 1000000)

                if not len(mirdeep2_dict[preMi]["mir5pID"]) == 0:
                      if perfectOffset_RPMdict.has_key(mirdeep2_dict[preMi]["mir5pID"]):
perfectOffset_RPMdict[mirdeep2_dict[preMi]["mir5pID"]] += (miRclass.perfect_offset_5p_count() /
float(totalReadCount) * 1000000)
                      else: perfectOffset_RPMdict[mirdeep2_dict[preMi]["mir5pID"]] = 0;
perfectOffset_RPMdict[mirdeep2_dict[preMi]["mir5pID"]] += (miRclass.perfect_offset_5p_count() /
float(totalReadCount) * 1000000)
                if not len(mirdeep2_dict[preMi]["mir3pID"]) == 0:
                      if perfectOffset_RPMdict.has_key(mirdeep2_dict[preMi]["mir3pID"]):
perfectOffset_RPMdict[mirdeep2_dict[preMi]["mir3pID"]] += (miRclass.perfect_offset_3p_count() /
float(totalReadCount) * 1000000)
                      else: perfectOffset_RPMdict[mirdeep2_dict[preMi]["mir3pID"]] = 0;
perfectOffset_RPMdict[mirdeep2_dict[preMi]["mir3pID"]] += (miRclass.perfect_offset_3p_count() /
float(totalReadCount) * 1000000)
                if not len(mirdeep2_dict[preMi]["mirID"]) == 0:
                      if perfectOffset_RPMdict.has_key(mirdeep2_dict[preMi]["mirID"]):
perfectOffset_RPMdict[mirdeep2_dict[preMi]["mirID"]] += ((miRclass.perfect_offset_M_5p_count()
+miRclass.perfect_offset_M_3p_count()) / float(totalReadCount) * 1000000)
                      else: perfectOffset_RPMdict[mirdeep2_dict[preMi]["mirID"]] = 0;
perfectOffset_RPMdict[mirdeep2_dict[preMi]["mirID"]] += ((miRclass.perfect_offset_M_5p_count()
+miRclass.perfect_offset_M_3p_count()) / float(totalReadCount) * 1000000)


      ## onset 중에 perfect (mismatch=0)로 mapping 되는 read 의 값을 RPM 값으로 출력함
      onsetPerfectFile = open(outputDir + "/" +outputFileName + "_OnsetPerfect_RPM.txt", 'w')
      for mi in perfectOnset_RPMdict.keys():
            onsetPerfectFile.write(mi + "\t" + "%.2f" % perfectOnset_RPMdict[mi] + "\t" +
`perfectOnset_ReadDict[mi]` + "\t" + `totalReadCount` + "\n")
      onsetPerfectFile.close()


      ## perfect (mismatch=0)로 mapping 되는 onset 과 offset 의 발현량 (RPM)과 fold change
(onset/offset)를 출력함
      onOffrateDic = dict()
      for mi in perfectOnset_RPMdict.keys():
```

```
            if perfectOffset_RPMdict.has_key(mi):
                  if perfectOffset_RPMdict[mi] ==0:
                        ratio = perfectOnset_RPMdict[mi] / min(filter(lambda x: x > 0,
perfectOffset_RPMdict.values()))
                        onOffrateDic[mi] = ratio
                  else:
                        ratio = perfectOnset_RPMdict[mi] / perfectOffset_RPMdict[mi]
                        onOffrateDic[mi] = ratio
      compareOnsetOffset = open(outputDir + "/" +outputFileName + "_OnsetOffset_Perfect_rate.txt",
'w')
      for mi in onOffrateDic.keys():
            compareOnsetOffset.write(mi + "\t" + "%.2f" % perfectOnset_RPMdict[mi] + "\t" +
"%.2f" % perfectOffset_RPMdict[mi] + "\t" + "%.2f" % onOffrateDic[mi] + "\t" + `totalReadCount` +
"\n")
      compareOnsetOffset.close()

      ## onset 으로 match 되는 read 를 perfect 와 mismatch 그룹으로 나누어서 각각의 발현량 (RPM)과 fold change
(perfect/mismatch)를 출력함
      perfectMisRateDic = dict()
      for mi in perfectOnset_RPMdict.keys():
            if mismatchOnset_RPMdict.has_key(mi):
                  if mismatchOnset_RPMdict[mi] ==0:
                        ratio = perfectOnset_RPMdict[mi] / min(filter(lambda x: x > 0,
mismatchOnset_RPMdict.values()))
                        perfectMisRateDic[mi] = ratio
                  else:
                        ratio = perfectOnset_RPMdict[mi] / mismatchOnset_RPMdict[mi]
                        perfectMisRateDic[mi] = ratio
      comparePerfectMis = open(outputDir + "/" +outputFileName + "_PerfectMismatch_onset_rate.txt",
'w')
      for mi in perfectMisRateDic.keys():
            comparePerfectMis.write(mi + "\t" + "%.2f" % perfectOnset_RPMdict[mi] + "\t" + "%.2f" %
mismatchOnset_RPMdict[mi] + "\t" + "%.2f" % perfectMisRateDic[mi] + "\t" + `totalReadCount` + "\n")
      comparePerfectMis.close()

      ## miRNA 의 arm-switching 을 보기 위해 miR-5p 와 miR-3p 의 발현량과 fold change(5p/3p)를 출력함
      perfectOnset5p3p_RPM = dict()
      preMi_matureMi = dict()
      for preMi in mirdeep2_dict.keys():
            if not len(mirdeep2_dict[preMi]["mir5pID"]) == 0:
                  if not len(mirdeep2_dict[preMi]["mir3pID"]) == 0:
                        mir5p = mirdeep2_dict[preMi]["mir5pID"]
                        mir3p = mirdeep2_dict[preMi]["mir3pID"]
                        preMi_matureMi[preMi] = [mir5p, mir3p]
                        if perfectOnset_RPMdict.has_key(mir5p) and
perfectOnset_RPMdict.has_key(mir3p):
                              if perfectOnset_RPMdict[mir3p] == 0:
                                    mir5p_mir3p_value = perfectOnset_RPMdict[mir5p] /
min(filter(lambda x: x > 0, perfectOnset_RPMdict.values()))
                              else: mir5p_mir3p_value = perfectOnset_RPMdict[mir5p] /
perfectOnset_RPMdict[mir3p]
                              perfectOnset5p3p_RPM[preMi] = mir5p_mir3p_value
      compare5p3p = open(outputDir + "/" +outputFileName + "_5p3p_perfectOnset_rate.txt", 'w')
      for preMi in perfectOnset5p3p_RPM.keys():
            mir5p = preMi_matureMi[preMi][0]
            mir3p = preMi_matureMi[preMi][1]
            compare5p3p.write(preMi + "\t" + mir5p + "\t" + "%.2f" % perfectOnset_RPMdict[mir5p] +
"\t" + mir3p + "\t" + "%.2f" % perfectOnset_RPMdict[mir3p] + "\t" + "%.2f" %
perfectOnset5p3p_RPM[preMi] + "\t" + `totalReadCount`+ "\n")
      compare5p3p.close()


def write_mismatchFile(mirdeep2_dict, preMi_monoDic, preMi_diDic, outputDir, outputFileName):
      ## 각 miRNA 별로 3'end 의 mismatch 를 보여주는 출력파일을 만드는 함수
      mismatchFile = open(outputDir + "/" + outputFileName + "_tpEnd_mismatch.txt", 'w')
      monoDic = dict()
      diDic = dict()
      for preMi in mirdeep2_dict.keys():
            if not (len(preMi_monoDic[preMi].keys())==0 and len(preMi_diDic[preMi].keys())==0):
                  mismatchFile.write(">"+ preMi + "\n")
                  for mono in preMi_monoDic[preMi].keys():
                        mismatchFile.write(mono + ":\t" + `preMi_monoDic[preMi][mono]` + "\n")
                        if monoDic.has_key(mono): monoDic[mono] += preMi_monoDic[preMi][mono]
                        else: monoDic[mono] = 0; monoDic[mono] += preMi_monoDic[preMi][mono]
```

```
                  for di in preMi_diDic[preMi].keys():
                        mismatchFile.write(di + ":\t" + `preMi_diDic[preMi][di]` + "\n")
                        if diDic.has_key(di): diDic[di] += preMi_diDic[preMi][di]
                        else: diDic[di] = 0; diDic[di] += preMi_diDic[preMi][di]
      mismatchFile.write(">Total proportion of mismatch on 3'end" + "\n")
      for total_mono in monoDic.keys():
            mismatchFile.write(total_mono + ":\t" + `monoDic[total_mono]` + "\n")
      for total_di in diDic.keys():
            mismatchFile.write(total_di + ":\t" + `diDic[total_di]` + "\n")
      mismatchFile.close()


def get_mirdeep2Data(mirdeep2File, outputDir, outputFileName, pre_annoDic, mature_annoDic,
species):
      ## miRdeep2 quantifier의 출력파일을 이용해서 데이터를 처리
      ## 1. pre-miRNA 별로 total read 수와 miR-5p와 miR-3p에 발현되는 read 수를 정리
      ## 2. pre-miRNA의 sequence와 sequence 내의 5p, 3p의 위치 정보를 저장
      ## 3. miRclass를 이용해서 alignment된 read에 생물학적인 정보 (5'heterogeneity, mismatch 등)을 통해
그룹을 재정의
      filein = open(mirdeep2File)
      mirdeep2_dict = dict()
      totalReadCount = 0
      for lines in filein.readlines():
            if lines.startswith(">" + species):
                  preMi = lines.strip()[1:]
                  switch = "on"
                  if not (mirdeep2_dict.has_key(preMi)): mirdeep2_dict[preMi] = {"mir5pID":[],
"mir3pID":[], "mirID":[], "exp":[], "seq":[]}
            if switch == "on":
                  if lines.startswith("total"):
                        readCountLine = lines.strip().split(" ")[-1]
                        if readCountLine == "count": readCount = 0
                        else: readCount = int(lines.strip().split(" ")[-1])
                        totalReadCount += readCount
                  if lines.startswith(species):
                        if lines.split(" ")[0][-2:] == "5p": mir5p = lines.split(" ")[0];
mirdeep2_dict[preMi]["mir5pID"] = mir5p
                        elif lines.split(" ")[0][-2:] == "3p": mir3p = lines.split(" ")[0];
mirdeep2_dict[preMi]["mir3pID"] = mir3p
                        else: mir = lines.split(" ")[0]; mirdeep2_dict[preMi]["mirID"] = mir
                  if lines.startswith("exp"):
                              exp = lines.strip().split(" ")[-1]
                              mirdeep2_dict[preMi]["exp"] = exp
                  if lines.startswith("seq"):
                        #print lines
                        seqLine_dict = dict()
                        count = lines.split("_")[2][1:]
                        misMatch = lines.strip().split("\t")[1]
                        seqID = lines.split(" ")[0] + "_mm" + misMatch
                        seq = lines.split(" ")[-1].split("\t")[0]
                        seqLine_dict[seqID] = seq
                        mirdeep2_dict[preMi]["seq"].append(seqLine_dict)
            if (len(lines.split(" ")) == 1 and not(lines.startswith(">"))): switch == "off"

      preMi_seqDic = dict()
      preMi_monoDic = dict()
      preMi_diDic = dict()
      for preMi in mirdeep2_dict.keys():
            miRclass = miRNAclass(mirdeep2_dict[preMi]["exp"], mirdeep2_dict[preMi]["seq"])
            mirdeep2_dict[preMi]["miRclass"] = miRclass
            preMi_seqDic[preMi] = miRclass.classifiedSequenceList()
            preMi_monoDic[preMi] = miRclass.mismatch_mono()
            preMi_diDic[preMi] = miRclass.mismatch_di()

      write_classFile(mirdeep2_dict, preMi_seqDic, outputDir, outputFileName)
      write_mismatchFile(mirdeep2_dict, preMi_monoDic, preMi_diDic, outputDir, outputFileName)
      write_bedFile(pre_annoDic, mature_annoDic, mirdeep2_dict, outputDir, outputFileName,
totalReadCount)
      write_statisticFile(mirdeep2_dict, preMi_seqDic, outputDir, outputFileName, totalReadCount)

pre_annoDic, mature_annoDic = getgff3(gff3FileName)
get_mirdeep2Data(mirdeep2File, outputDir, outputFileName, pre_annoDic, mature_annoDic, species)
```