Elementi di bioinformatica

Moretti Simone

24/25-tema2

Parametri in input

Ho eseguito uno script eseguile da linea di comando che riceve 3 parametri in input (facoltativi con valori di default):

- --fileGtf (-g) seguito dal percorso del file gtf
- --fileBam (-b) seguito dal percorso del file bam
- --output (-o) seguito dal nome che si vuole dare al file in output

I valori di default sono rispettivamente:

- ./annotation_one_tr_chr21.gtf
- ./sample-chr21.bam
- spliced_reads

Librerie utilizzate

- argparse: per ricevere i parametri in input
- pysam: importando AlignmentFile per leggere e manipolare il file bam e i singoli reads
- pandas: per memorizzare i record del file gtf in un data frame e quindi consultarli più facilmente
- re: per trovare i valori di ‘transcript_id’ e ‘gene_id’ all’interno del campo ‘attributes’ del data frame
- Bio: importando Seq e SeqRecord per creare delle sequenze annotate e formattarle in fastq

Descrizione e commenti

La prima parte prende i dati in input usando argparse, li legge e manipola il data frame per renderlo più facilmente utilizzabile

```
parser = argparse.ArgumentParser()
parser.add_argument("-g", "--fileGtf", default = "./annotation_one_tr_chr21.gtf", type = argparse.FileType('r'), help="Nome del file gtf")
parser.add_argument("-b", "--fileBam", default= "./sample-chr21.bam", type = argparse.FileType('rb'), help="Nome del file Bam")
parser.add_argument("-o", "--output", default = "spliced_reads", type = str, help="Nome del file di output")
args = parser.parse_args()

input_bam = args.fileBam
pysam.index(input_bam.name)
bam_file = AlignmentFile(input_bam, "rb")

input_gtf = args.fileGtf
df = pd.read_csv(input_gtf, sep="\t", header=None)
replace_dict = {0 : 'reference', 1 : 'source', 2 : 'feature', 3 : 'start',
                4 : 'end', 5 : 'score', 6 : 'strand', 7 : 'frame', 8 : 'attributes'}
df.rename(columns = replace_dict, inplace = True)
df["transcript_id"] = df["attributes"].apply(lambda x : re.search(r"transcript_id\s"+ "(.+?);", x).group(1))
df["gene_id"] = df["attributes"].apply(lambda x : re.search(r"gene_id\s" + "(.+?);", x).group(1))

if bam_file.count() == 0:
    print("il file bam non contiene reads")
    exit(0)

if (bam_file.nreferences > 1):
    print("il file bam contiene più di una reference")
elif (bam_file.mapped == 0):
    print("il file non contiene reads mappati")
elif len(df.reference.unique()) > 1:
    print("il file gtf contiene più di una reference")
else:
    if str(df.reference.unique()[0]) != str(bam_file.references[0]):
        print("la reference del file gtf potrebbe non corrispondere con quella del file bam")
```

Inoltre effettua dei controlli sui file in input, se il file bam non contiene read ferma lo script, se il file bam o il file gtf hanno più di una reference, o il bam non ha reads mappati allora il programma esegue solo al seconda parte (creazione del fastq)

```
    if str(df.reference.unique()[0]) != str(bam_file.references[0]):
        print("la reference del file gtf potrebbe non corrispondere con quella del file bam")

    start_mapped_position = float("inf")
    end_mapped_position = float("-inf")
    for read in bam_file.fetch():
        if read.is_mapped:
            start_mapped_position = min(start_mapped_position, read.reference_start)
            end_mapped_position = max(end_mapped_position, read.reference_end)
    start_mapped_position += 1
    end_mapped_position += 1
    print("tutti i reads nel file bam sono mappati tra le posizioni " + str(start_mapped_position)
          + " e " + str(end_mapped_position) + " della reference")

    df_grouped = df[["start", "end", "transcript_id", "gene_id"]].groupby(["transcript_id","gene_id"]).agg({"start": "min", "end": "max"})
    df_grouped.reset_index(inplace = True)

    max_position_gtf = max(df_grouped.end)
    min_position_gtf = min(df_grouped.start)
```

Se il nome della reference del file bam e quella del file gtf sono diverse scrive solo un avviso ma esegue ugualmente tutto il programma, questo perché considero che due nomi diversi potrebbero comunque indicare lo stesso riferimento (ad esempio 21 e chr21).

Trova il range di basi sul riferimento dentro il quale si posizionano tutti i reads (+1 per rendere la posizione 1-based come nel file gtf), e raggruppa per trancript_id (e gene_id dato che servirà dopo), aggregando start con l'operatore min ed end con l'operatore max, crea quindi un data frame dove per ogni trascritto ha il range di posizioni che occupa quel trascritto sul riferimento.

Trova anche il range di posizioni che copre l'intero gtf.

```
60      if start_mapped_position < min_position_gtf or end_mapped_position > max_position_gtf:
61          print("i reads sono stati sequenziati fuori dal range coperto dal file gtf")
62      else:
63          df_grouped = df_grouped[(df_grouped.start <= start_mapped_position) & (df_grouped.end >= end_mapped_position)]
64
65          if df_grouped.empty:
66              print("i reads sono stati sequenziati da trascritti diversi")
67          else:
68              transcript_possibili = list(zip(df_grouped.transcript_id, df_grouped.gene_id))
```

Fa altri controlli, se un read è stato mappato fuori dal range coperto dal file gtf allora crea solo il fastq, altrimenti con una maschera mantiene solo i trascritti che "inglobano" la regione mappata dai reads.

Se il data frame risultante è vuoto vuol dire che non c'è un unico trascritto da cui sono stati ricavati tutti i reads, e quindi che sono stati ricavati da più trascritti.

A questo punto nel data frame (raggruppato) ho pensato che potrebbe esserci più di un trascritto, ad esempio la stessa zona occupata da un trascritto sullo strand forward è occupata da un trascritto diverso sullo strand reverse.
Il problema era come capire da quale strand provenissero i reads. Cercando su internet ho trovato che alcuni metodi di sequenziamento non mantengono l'informazione dello strand dal quale vengono ricavati i reads.
I reads nei file hanno strand diversi ma sono tutti paired end, quindi non si può escludere a prescindere che non siano stati sequenziati dallo stesso trascritto, Tuttavia controllando lo strand di tutti i read1 delle coppie ho notato che non tutti gli strand sono concordi, ho quindi immaginato che questi reads siano stati sequenziati perdendo l'informazione dello strand di appartenenza.
Data questa supposizione ho cercato un metodo generale per risolvere il problema sfruttando l'informazione degli introni.

```
68          transcript_possibili = list(zip(df_grouped.transcript_id, df_grouped.gene_id))
69
70          if len(transcript_possibili) > 1:
71              df = df[(df.end >= start_mapped_position) & (df.start <= end_mapped_position) & (df.feature == "exon")]
72              df.reset_index(inplace = True)
73              introns = sorted(bam_file.find_introns(bam_file.fetch()))
74              for (transcript_id, _) in list(transcript_possibili):
75                  if len(df[transcript_id == df["transcript_id"]]) != len(introns)+1:
76                      transcript_possibili.remove((transcript_id, _))
77                  else:
78                      for (i, (start_intron, end_intron)) in enumerate(introns, start=df[df["transcript_id"] == transcript_id].index[0]):
79                          if (df.loc[i].end != start_intron or df.loc[i+1].start != end_intron+1):
80                              transcript_possibili.remove((transcript_id, _))
81                              break
82
83          print("i reads sono stati sequenziati dal trascritto " + transcript_possibili[0][0] + " del gene " + transcript_possibili[0][1])
```

Prima di tutto salva tutti i trascritti (e i geni corrispondenti) rimasti nel data frame raggruppato in una lista (questa sarà la lista che contiene tutti i possibili trascritti, nei casi di esempio a questo punto la lista ha già lunghezza 1).
Se c'è più di un trascritto nella lista dei trascritti possibili prende il data frame originale (non quello raggruppato) e tiene solo gli esoni che hanno un overlap con la regione in cui sono mappati i reads.
Ricava la lista degli introni supportati dal file bam, la ordina e la usa per capire da quale trascritto sono stati sequenziati i reads.

Prima di tutto rimuove i trascritti che hanno un numero di esoni non concorde con quello degli introni (dato che un introne si trova tra due esoni il numero di introni deve essere 1 in meno rispetto a quello degli esoni),
per ogni trascritto che rispetta questa condizione scorre gli introni e vede se si "incastrano" tra gli esoni correttamente, se non succede allora scarta il trascritto.
Alla fine nella lista dei trascritti possibili rimane solo una tupla (trascritto, gene).
Tuttavia questo metodo funziona solo se non ci sono errori nei reads tali che vengano trovati degli introni sbagliati, e se nel file gtf ci sono tutti gli esoni dei trascritti presi in considerazioni (anche questi devono essere giusti), nella pratica non so quanto queste condizioni vengano effettivamente rispettate.

```
85  with open(args.output + ".fastq", "w") as output_file:
86      for read in bam_file.fetch():
87          if "N" in read.cigarstring:
88              qualities = read.query_qualities if read.query_qualities != None else [0]*len(read.query_sequence)
89              record = SeqRecord(Seq(read.query_sequence), id=read.query_name,
90                                 description = "base " + str(read.next_reference_start) + " on " + read.reference_name,
91                                 letter_annotations={"phred_quality": qualities})
92              output_file.write(format(record,"fastq"))
```

Nell'ultima parte salva su un file fastq i soli reads spliced, che quindi hanno un gap da introne (rappresentato da 'N' nella cigarstring), per farlo prende la lista di qualities del read se questo ce l'ha, altrimenti ne imposta una di default (tutte le qualità a 0) e crea un oggetto SeqRecord che viene formattato in fastq e scritto su un file con il nome passato in input (con l'aggiunta dell'estensione fastq).

Un file fastq è un file di puro testo usato per memorizzare la sequenza primaria di un read e una sequenza di indici di qualità, che, per ogni base, indica la probabilità che la base sia corretta.

Ogni read viene memorizzato in 4 righe, la prima è composta da una @ seguita dall'identificatore del read e, tipicamente, da una descrizione (solitamente informazioni relative al sequenziamento, ad esempio lo stumento usato) nel mio caso ho inserito come descrizione la posizione del read sulla reference (in assenza di altre informazioni e per non lasciare lo spazio vuoto). La seconda riga contiene la sequenza primaria del read, la terza riga contiene un + opzionalmente seguito nuovamente dall'identificatore del read e dalla descrizione infine la quarta riga contiene la sequenza degli indici di qualità (o phred value).
Gli indici (Q) vengono memorizzati come caratteri ASCII ottenuti tramite la formula Q = ASCII(min(Q, 93)+33).