project_turnin

June 6, 2024

[ ]:
```
import gzip
from Bio import SeqIO
from Bio.Seq import Seq
from Bio.SeqRecord import SeqRecord
from collections import defaultdict, Counter
from collections import defaultdict
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

[ ]:
```
# Function to build an index from the transcriptome data
# Map k-mers to transcript IDs
def build_index(transcriptome_file, k):
    # Empty dictionary to store index (hash-table)
    index = defaultdict(list) #Get list of transcripts that contain a given␣
 ↪k-mer

    # get transcriptome
    with open(transcriptome_file, "r") as handle:
        # Parse each sequence in file
        for record in SeqIO.parse(handle, "fasta"):

            # Convert sequence to string
            seq = str(record.seq)
            # Get transcript ID
            transcript_id = record.id

            # Get all possible k-mers of length k
            for i in range(len(seq) - k + 1):
                # Extract k-mer at position i, with length k
                kmer = seq[i:i+k]

                # Add transcript ID to hash-table for this k-mer
                index[kmer].append(transcript_id)
                    #ex ATC -> [isoform1, isoform2]

    return index
```

[ ]:
```python
# Function to compute the reverse complement of a sequence
def reverse_complement(seq):
    # Dictionary mapping each nucleotide to complement
    complement = {
        'A': 'T',  # Adenine -> Thymine
        'C': 'G',  # Cytosine -> Guanine
        'G': 'C',  # Guanine -> Cytosine
        'T': 'A',  # Thymine -> Adenine
        'N': 'N'   # Any base -> Any base (gets dealt with somewhere else)
    }

    # Empty string to store reverse complement sequence
    reverse_comp_seq = ''

    # Loop through sequence in reverse order
    for base in reversed(seq):
        # Add complement of current base to reverse complement sequence
        reverse_comp_seq += complement[base]

    return reverse_comp_seq

def generate_kmers(seq, k):
    kmers = set()  # Set to store unique k-mers

    # Generate k-mers from the sequence
    for i in range(len(seq) - k + 1):
        kmer = seq[i:i + k]
        if 'N' not in kmer:  # Skip k-mers with 'N'
            kmers.add(kmer)

    return kmers
```

[ ]:
```python
def pseudoalign(index, reads_file, k):
    equivalence_classes = defaultdict(int)  # Dictionary to store equivalence␣
 ↪classes and their counts

    with open(reads_file, "rt") as handle:  # Open the reads file
        for record in SeqIO.parse(handle, "fasta"):  # For each sequence in the␣
 ↪reads file
            read_seq = str(record.seq)  # Convert sequence to string
            rc_read_seq = reverse_complement(read_seq)  # Reverse complement of␣
 ↪the sequence

            matched_transcripts = None  # Initialize matched transcripts as None

            # Generate k-mers from read (forward strand)
            kmers_read = generate_kmers(read_seq, k)
```

```
            # Generate k-mers from reverse complement of the read
            kmers_rc_read = generate_kmers(rc_read_seq, k)

            # Get matches from forward sequence
            for kmer in kmers_read:
                if kmer in index:
                    if matched_transcripts is None: # If no matches
                        matched_transcripts = set(index[kmer]) # Initialize set␣
↪of matched transcripts
                    else: # If there are matches
                        matched_transcripts.intersection_update(index[kmer]) #␣
↪Update set of matched transcripts

            # Get matches from reverse complement sequence
            for kmer in kmers_rc_read:
                if kmer in index:
                    if matched_transcripts is None:
                        matched_transcripts = set(index[kmer])
                    else:
                        matched_transcripts.intersection_update(index[kmer])

            #print(f"Matched transcripts: {matched_transcripts}")

            if matched_transcripts and len(matched_transcripts) > 0:  # If␣
↪there are matches, create an equivalence class
                eq_class = tuple(sorted(matched_transcripts))
            else:  # If no matches, then 'NA' as equivalence class
                eq_class = ('NA',)

            # Update count of THIS equivalence class
            equivalence_classes[eq_class] += 1

    return equivalence_classes
```

[ ]:
```
# Main function to return results
def run_pseudoalignment(transcriptome_file, reads_file, k):
    index = build_index(transcriptome_file, k) #Get index made of k-mers length␣
↪k
    pseudoalignment_results = pseudoalign(index, reads_file, k) #Pseudoalign

    # Output results
    for eq_class, count in pseudoalignment_results.items():
        num_items = len(eq_class) # Number of isoforms in equivalence class
        isoforms = ",".join(eq_class) #Join isoforms into single string
        if not isoforms: #If no isoforms, set to NA
            isoforms = 'NA'
```

```
        # print(f"{count}\t{num_items}\t{isoforms}") #Print count, # items, and␣
↪isoforms

    # statistics
    eq_class_sizes = {} #Empty dictionary to count sizes of equivalence classes

    # Loop through each equivalence class in pseudoalignment results
    for eq_class in pseudoalignment_results:
        # size of current equivalence class
        size = len(eq_class)

        # If size already in dictionary, increment its count
        if size in eq_class_sizes:
            eq_class_sizes[size] += 1
        # If size is not in dictionary, add it with a count of 1
        else:
            eq_class_sizes[size] = 1

    # Loop through each size and its count in eq_class_sizes dictionary
    # for size, count in eq_class_sizes.items():
    #     # Print size of equivalence class and number of classes with that size
    #     print("Equivalence class size", size, ":", count, "classes")

    return pseudoalignment_results
```

[ ]:
```
def equivalence_classes_to_dataframe(equivalence_classes):
    data = {
        "counts": [],
        "number of items in equivalence class": [],
        "isoforms in equivalence class": []
    } #Empty dictionary to store data

    for eq_class, count in equivalence_classes.items(): #Loop through each␣
↪equivalence class
        num_items = len(eq_class)
        isoforms = ",".join(eq_class) if eq_class else "NA" #Join isoforms into␣
↪single string

        data["counts"].append(count) #Append count
        data["number of items in equivalence class"].append(num_items) #Append␣
↪number of items
        data["isoforms in equivalence class"].append(isoforms) #Append isoforms

    df = pd.DataFrame(data)
    df = df.sort_values(by="counts", ascending=False) #Sort by descending order
    return df
```

[ ]:
```
# Parameters
transcriptome_file = "/Users/timothyliu/Documents/121/Project/
 ↪chr11_transcriptome.fasta"
reads_file = "/Users/timothyliu/Documents/121/Project/reads.fasta"
sam_file = "aligned_reads.sam"
k = 31  # k-mer length

# Run pseudoalignment implementation
pseudoalignment_results = run_pseudoalignment(transcriptome_file, reads_file, k)
```

[ ]:
```
df = equivalence_classes_to_dataframe(pseudoalignment_results)
df.to_csv('/Users/timothyliu/Documents/121/Project/output.csv', index=False)
df.head(15)
```

[ ]:

	counts	number of items in equivalence class	\
32	66897	1	
4302	61263	2	
1737	15353	1	
2162	12320	7	
8907	12097	1	
3871	10440	5	
547	10038	3	
1989	9317	2	
9590	8807	2	
9885	7457	8	
9861	7386	1	
9589	6755	3	
2161	6028	11	
9594	5954	1	
1465	5545	7	

	isoforms in equivalence class
32	NA
4302	ENST00000329251,ENST00000496634
1737	ENST00000536684
2162	ENST00000227157,ENST00000379412,ENST0000039622…
8907	ENST00000393067
3871	ENST00000345732,ENST00000389939,ENST0000053207…
547	ENST00000321153,ENST00000530398,ENST00000530797
1989	ENST00000228140,ENST00000525634
9590	ENST00000527673,ENST00000532567
9885	ENST00000227378,ENST00000524552,ENST0000052611…
9861	ENST00000260197
9589	ENST00000527673,ENST00000527791,ENST00000532567
2161	ENST00000227157,ENST00000379412,ENST0000039622…
9594	ENST00000527673
1465	ENST00000314138,ENST00000524496,ENST0000052656…

[ ]:
```
df.describe(include='all').T
```

[ ]:
```
                                        count unique  top freq        mean  \
counts                                10489.0    NaN  NaN  NaN  122.273429
number of items in equivalence class  10489.0    NaN  NaN  NaN    5.079321
isoforms in equivalence class           10489  10489   NA    1         NaN

                                              std  min  25%   50%   75%  \
counts                                1007.425181  1.0  2.0  10.0  46.0
number of items in equivalence class     5.316185  1.0  2.0   4.0   7.0
isoforms in equivalence class                 NaN  NaN  NaN   NaN   NaN

                                         max
counts                               66897.0
number of items in equivalence class    54.0
isoforms in equivalence class            NaN
```