Palma Alessio - palma.1837493@studenti.uniroma1.it
Zannini Simone - zannini.1837364@studenti.uniroma1.it

Progetto Biologia Computazionale

Il progetto consiste in un tool da riga di comando che permette all'utente di effettuare query all'SRA (Sequence Read Archive), ottenerne i risultati formattati come file XML e, successivamente, filtrare i risultati sugli attributi e i metadati desiderati. Il tool produce due tabelle che sono collegate a livello di filtering.
A differenza del sito dell'NCBI, che permette una visualizzazione di tali dati raggruppandoli per esperimento, il nostro tool li organizza per ID del relativo BioProject.

1. Tecnologie utilizzate e flusso d'esecuzione

Il tool è composto da due eseguibili: parser.py e filter.py. Il parser prevede due possibili modalità d'esecuzione:

- La prima (modalità query) consente di effettuare automaticamente la query d'interesse tramite le utilities delle EDirect, e successivamente verrà "parsato" il file XML risultante dalla query per produrre le tabelle di output, con le informazioni raggruppate per BioProject;
- La seconda (modalità file locale), prevede che l'utente abbia già un file XML prodotto tramite `efetch -format xml`. L'output sarà poi identico alla modalità query.

Flusso d'esecuzione per la modalità query:

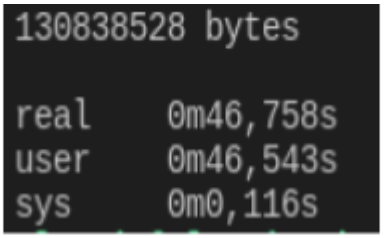
1. Il tool parser.py viene avviato tramite riga di comando con il parametro `-q`;
2. Viene chiamato il file downloaderRunner.py, il quale produce uno script di bash downloader.sh che usa le edirect per scaricare la query in formato XML. Tale script verrà mandato in esecuzione tramite il comando `sbatch --quiet --wait downloader.sh`, si presuppone quindi l'uso su un sistema con scheduler Slurm e dotato di EDirect, come ad esempio le macchine del Cineca. Il parametro `--quiet` serve semplicemente a non produrre nulla sullo standard output, mentre il parametro `--wait` serve ad attendere la terminazione del comando prima di proseguire nel codice python. Lo script si occupa anche di aggiungere un unico elemento radice al file XML;
3. A questo punto quando la query sarà terminata verrà prodotto dall'efetch un file XML, da qui riprenderà l'esecuzione del codice Python, che "parserà" l'XML sfruttando la libreria xml.dom.pulldom;
4. Dopo aver processato ogni experiment dall'XML, vengono salvati in un file di backend tutti i dati organizzati per run, in modo che da poter ricostruire le relazioni tra gli oggetti dal tool di filtering;
5. Finito il parsing, vengono prodotti i file di output in formato csv.

Flusso d'esecuzione per la modalità file locale:

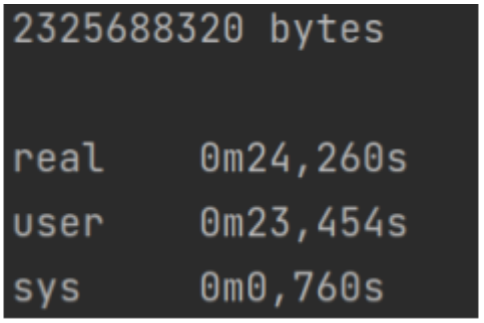
1. Il tool parser.py viene avviato tramite riga di comando con il parametro `-i`, che specifica quale file XML viene dato in input allo script Python per il parsing.
2. Il tool aggiunge un unico elemento radice al file XML dato in input, per evitare di trovarsi di fronte ad XML mal formattati;
3. Il codice Python "parsa" l'XML dato in input sfruttando la libreria xml.dom.pulldom;

4. Dopo aver processato ogni experiment dall'XML, vengono salvati in un file di backend tutti i dati organizzati per run, in modo che da poter ricostruire le relazioni tra gli oggetti dal tool di filtering;
5. Finito il parsing, vengono prodotti i file di output in formato csv e viene riportata su terminale la dimensione totale del dataset. La seconda tabella avrà sempre le colonne ordinate per popolarità.

La scelta è ricaduta sulla libreria xml.dom.pulldom perché è un parser che permette di scorrere il documento XML un elemento alla volta, e creare il DOM radicato in un elemento solo quando richiesto dal programmatore, occupando così pochissima memoria.
In prima istanza la scelta si era riversata sulla libreria xml.dom.minidom sia per la facilità di utilizzo, sia perché produceva i risultati richiesti in maniera abbastanza rapida. L'utilizzo di memoria di questa libreria era però fortemente limitante anche per macchine con ampia disponibilità di RAM, in quanto, come possibile vedere seguentemente, occupava molta più memoria di quanto fosse necessario considerate le dimensioni del file. Questo perchè xml.dom.minidom crea direttamente il DOM dell'intero file XML. Il trade off quindi raggiunto con l'utilizzo di xml.dom.pulldom è tra velocità di esecuzione (la libreria scelta è nell'ordine del doppio del tempo rispetto a xml.dom.minidom) e l'occupazione di memoria (un'ordine di grandezza in meno).



A fronte di un file di input di 152 MB, la versione del tool con pulldom usa 130838528 bytes = 0,130 GB di memoria e ci impiega 46 secondi.



Si noti come con minidom il tempo di esecuzione sia circa la metà a fronte dello stesso file in input, mentre la memoria occupata sia ≅ 2.3GB.

Flusso d'esecuzione per il tool filter.py:

1. Il tool legge il file di backend passato in input per ricostruire le strutture che permettono di determinare le associazioni tra BioProject, Experiment, BioSample e Run. In questo modo sapremo quali sono gli attributi e i rispettivi dati sui quali l'utente potrà applicare i suoi filtri. Viene riportata su terminale la dimensione totale del dataset.

2. All'utente verrà fornita la possibilità di filtrare su due macro insiemi di filtri, uno statico (prima tabella) e uno dinamico (seconda tabella). L'attività di filtering potrà essere effettuata grazie all'ausilio di una semplice interfaccia su terminale. Il sistema di filtraggio è completamente interattivo, in quanto aggiorna costantemente sia le possibili scelte in base ai filtri selezionati sia le tabelle stesse che vengono generate a ogni scelta dell'utente. A ogni scrittura dei file durante il filtraggio viene riportata la grandezza attuale del dataset su terminale.
3. Filtrare su un qualsiasi attributo porterà all'eliminazione degli experiment (e quindi anche dei sample) che non rispettano i filtri inseriti, causando una scrematura nell'header della seconda tabella e un ricalcolo dei valori della prima.
4. Quando l'utente deciderà di terminare la sua attività di filtraggio, verranno generate le tabelle definitive. La seconda tabella avrà sempre gli attributi ordinati per popolarità dell'attributo nei vari progetti (cioè quante volte quel metadato appare in più progetti).

2. Utilizzo da riga di comando

parser.py in modalità query:

```
python3 parser.py -q "query" --o1 table1.csv --o2 table2.csv -b backend.csv
```

Il parametro `-q` prende tra virgolette la query che vogliamo fare all'SRA, `--o1` indica il path della prima tabella che verrà prodotta mentre `--o2` indica il path della seconda tabella. L'unico parametro opzionale è `-b` che indica il file dove salvare la struttura di backend, se tale parametro non viene fornito la struttura viene salvata in un file nascosto denominato ".backend.csv".

parser.py in modalità file locale:

```
python3 parser.py -i inputfile --o1 table1.csv --o2 table2.csv -b backend.csv
```

I parametri sono uguali al caso precedente, tranne `-i` che specifica il path dell'XML di input.

filter.py:

```
python3 filter.py -b backend.csv --o1 filtered_table1.csv --o2 filtered_table2.csv
```

I parametri `--o1` e `--o2` permettono di definire i path in cui verranno salvate le tabelle di output filtrate. L'unico parametro opzionale è `-b` che indica il file da cui leggere la struttura di backend, se tale parametro non viene fornito la struttura viene letta dal file nascosto denominato ".backend.csv".

Con il termine “**prima tabella**” ci riferiamo alla tabella con header:

`BioProject,Platforms,#SRA_experiments,#Runs,Size,Layouts,Sources,Strategies,Organisms,Links`

Con il termine “**seconda tabella**” ci riferiamo alla tabella che mostra i metadati contenuti in tutti i sample degli experiment di un BioProject. L’header è nella forma:
`BioProject,` m_1`,` m_2`, …,` m_n
Dove m_1, m_2, …, m_n è la lista dei metadati.

Per completezza, nel file utilities.py possiamo trovare tutte quelle funzioni che sono riutilizzate più volte all’interno del codice, con vari import essenziali al funzionamento del programma.

Attributes		
	body mass index	19.2
	body habitat	UBERON:feces
	body product	UBERON:feces
	tissue	UBERON:feces
	geographic location	USA
	dominant hand	I am right handed
	elevation	205.9
	environmental medium	feces
	environmental package	human-gut
	geographic location	USA:TN
	host subject id	b03f65bcc787da49a8b69d25a6eda0c7787baf47bd413
	race	Caucasian
	sample type	Stool
	sex	female
	acid_reflux	I do not have this condition
	acne_medication	No
	acne_medication_otc	No
	add_adhd	I do not have this condition
	age_cat	60s
	age_corrected	60
	age_years	60
	alcohol_consumption	Yes
	alcohol_frequency	Occasionally (1-2 times/week)
	antibiotic_history	I have not taken antibiotics in the past year.
	appendix_removed	Yes
	artificial_sweeteners	Never
	asd	I do not have this condition
	autoimmune	I do not have this condition
	birth_year	1958
	bmi_cat	Normal
	bmi_corrected	19.2
	bowel_movement_frequency	Less than one
	bowel_movement_quality	I tend to be constipated (have difficulty passing stool) -

Si noti come la lista dei metadati (Attributes nella foto) di un singolo experiment può essere anche molto lunga e esente da standard, dato che tali dati sono inseriti da esseri umani.

vioscreen_calcium_freq	Not provided
vioscreen_calcium_from_dairy_servings	Not provided
vioscreen_calcium_servings	Not provided
vioscreen_calories	Not provided
vioscreen_carbo	Not provided
vioscreen_cholest	Not provided
vioscreen_choline	Not provided
vioscreen_clac9t11	Not provided
vioscreen_clat10c12	Not provided
vioscreen_copper	Not provided
vioscreen_coumest	Not provided
vioscreen_cystine	Not provided
vioscreen_d_cheese	Not provided
vioscreen_d_milk	Not provided
vioscreen_d_tot_soym	Not provided
vioscreen_d_total	Not provided
vioscreen_d_yogurt	Not provided
vioscreen_daidzein	Not provided
vioscreen_database	Not provided
vioscreen_delttoco	Not provided
vioscreen_discfat_oil	Not provided

Non fornendo l'NCBI uno standard, purtroppo si verificano situazioni del genere. Questo BioSample in particolare aveva 273 valori "not provided".

3. Limiti delle EDirect e spunti per il futuro

Nel corso del progetto abbiamo incontrato diversi bug nell'utilizzo delle EDirect, tutti stati segnalati all'autore, che però al momento ne ha corretto soltanto uno.
Innanzitutto all'interno degli XML mancavano i tag relativi al `LIBRARY_LAYOUT`, ciò è stato risolto prontamente dall'autore in seguito ad una nostra email.
Il primo problema che lo sviluppatore non è riuscito a risolvere concerne lo scaricamento dell'XML di alcune query, capita che il download si fermi ad un punto imprecisato del documento in seguito all'errore:
`Unexpected characters after end element name ERROR: Unparsable XML element.`
Altri due problemi, entrambi riguardanti la formattazione dell'XML, sono stati segnalati e ancora non abbiamo ricevuto riscontro:

- Il primo consiste nella mal formattazione dei file XML in quanto manca un elemento radice, infatti al livello radice è presente una lista di tag `EXPERIMENT_PACKAGE_SET` e non un unico elemento che fa da padre a tutti. Noi ci siamo occupati di risolvere il problema a posteriori, facendo aggiungere automaticamente il tag mancante tramite riga di comando (come è possibile vedere nello script prodotto da downloaderRunner.py). Ciò chiaramente aggiunge un suo tempo di computazione in quanto richiede di reindirizzare le righe del file (che può essere anche di grandi dimensioni) successive alla terza. Sicuramente inserire questo tag quando si genera l'XML sarebbe molto più efficiente, ma purtroppo la generazione del file non è in nostro controllo;

- Il secondo consiste nel fatto che può capitare che l'XML prodotto abbia alcuni tag in cui gli attributi non sono separati da spazio, o siano mal formattati. Ad esempio alla riga 74125285 of cat.xml troviamo il seguente tag:
  ```
  <Member
  member_name=""accession="SRS4253767"sample_name="GSM3563849"sa
  mple_title="An-control_time-120_rep-C"spots="13313508"bases="1
  344664308"tax_id="264203"organismZymomonas mobilis subsp.
  mobilis ZM4="TCC 74125285 31821">.
  ```

Chiaramente uno qualunque di questi errori causa un crash di qualsiasi parser, perchè non si tratta di file ben formattati secondo lo standard XML.
Per il futuro risolvere questi bug porterebbe sicuramente un maggior numero di query interpretabili con successo dal nostro tool.
Anche i metadati dei sample, dopo il processamento, risultano molto poco leggibili in quanto come prima specificato scritti da umani. Nel nostro piccolo riteniamo che fornire uno standard per l'inserimento potrebbe conferire maggiore validità e robustezza all'attività di filtering e visualizzazione.