Laboratorio di biologia computazionale

AWK

AWK è un programma che ci permette di manipolare il testo ed è anche, quasi, un linguaggio di programmazione. I programmi di AWK hanno formato con accoppiamento pattern-azioni: l'azione di default è di stampare tutte le righe e un pattern può non esserci (non è indispensabile). È fondamentale conoscere l'input: se vogliamo maneggiare un file ed estrarne delle informazioni, dobbiamo sapere come queste informazioni sono salvate nel file di input; prima di cominciare ad analizzare il nostro programma in AWK, quindi, dobbiamo leggere il file da dare in input e in generale se usiamo in input un file standard non dovrebbero esserci problemi perché i formati standard si conoscono (ma dare un'occhiata con il comando less non richiede troppo tempo e da garanzie). Di default il delimitatore dei campi per ogni record o riga è il tab oppure uno o più spazi ma si può modificare con l'opzione -F (se non voglio dividere il mio file in colonne o record con il tab ma con il punto e virgola scriverò "awk -F ';' <hs_annotation.gtf '{print $1}' | less -S); ogni campo è definito con $field_number e $0 si riferisce a tutti i campi. I pattern possono essere espressioni, espressioni regolari e BEGIN e END.

- = assignment operator; sets a variable equal to a value or string
- == equality operator; returns TRUE is both sides are equal
- != inverse equality operator
- && logical AND
- || logical OR
- ! logical NOT
- <, >, <=, >= relational operators
- +, −, /, *, %, ^
- String concatenation

Per selezionare le righe utilizziamo una serie di operatori: l'= è un operatore di assegnazione e assegna una variabile a un valore o a una stringa; l'== è un operatore di uguaglianza e riporta "TRUE" se entrambe le parti sono uguali; al contrario funziona il != e gli operatori logici riportati in figura.

Le azioni poi possono essere diverse; abbiamo detto che l'azione di default è la stampa ma è possibile anche fare l'assegnazione a variabile, operazioni matematiche e l'incrementatore (a ogni riga aumentare il valore di un certo numero [{c=c+1}, {c +=1} oppure {c++}].

Un esempio di formato file molto comune è il file GTF che contiene: seqname (nome di cromosoma- con o senza prefisso chr- o scaffold); source (nome del programma che ha generato questa feature o la fonte dei dati); feature (caratteristica che definisce il nome come gene o variazione o similarità); start (posizioine di inizio di una feature con numeramento della sequenza che comincia con l'uno); fine (posizione finale della feature); score (un valore a virgola mobile); uno strand (definito per l'orientamento come positivo, +, o negativo, -); un frame (un valore tra 0 e 2 che indica da che nucleotide del codone si deve iniziare); attribute (una semicolonna che fornisce informazioni addizionali riguardo ogni caratteristica).

Allineamento

Il formato FASTA è quel formato file di testo che è formato da una riga di descrizione e una riga di sequenza. La riga di descrizione inizia con ">" e questo è il tratto distintivo della descrizione rispetto alla sequenza che è quella che segue la descrizione (possono essere nucleotidi o amminoacidi). All'interno di un file possiamo avere una o più sequenze (una successiva all'altra) distinte dal fatto che ogni sequenza è sempre preceduta dalla descrizione. Se io volessi, dato un file FASTA che contiene molte sequenze, tirare fuori solo le righe di ID o di descrizione potrei farlo cercando il ">" a inizio riga.

Allineamento globale, Needleman-Wunsch. Partiamo dai due file fasta "FOXS1.fasta" e "foxs1.fasta" che sono sequenze nucleotidiche da confrontare globalmente; i nomi dei geni umani devono essere scritti in maiuscolo mentre i geni murini si scrivono con iniziale maiuscola e il resto minuscolo. Per farlo sfruttiamo un software online che si chiama EMBOSS (EMBOSS Needle per quanto riguarda l'allineamento globale ed EMBOSS Water oer l'allineamento locale) dove possiamo semplicemente caricare le nostre due sequenze (visualizzate sulla bash con less) mantenere l'accoppiamento pair come output e chiedere al software il risultato (SUBMIT ma prima di SUBMIT abbiamo un pannello che ci propone più opzioni come l'utilizzo di

```
# Aligned_sequences: 2
# 1: EMBOSS_001
# 2: EMBOSS_001
# Matrix: EDNAFULL
# Gap_penalty: 10.0
# Extend_penalty: 0.5
#
# Length: 1364
# Identity:    1019/1364 (74.7%)
# Similarity:  1019/1364 (74.7%)
# Gaps:         115/1364 ( 8.4%)
# Score: 3787.5
```

matrici diverse o di diverse penalities per i gap). L'output accoppierà le due sequenze end to end e ci dirà quanto le due sequenze somiglino, i punteggi e l'identità totale, la percentuale di similarità e lo score. C'è differenza tra percentuale di identità e di similarità: in questo caso sono uguali perché i nucleotidi identificati sono gli stessi mentre nel caso degli allineamenti locali le percentuali possono variare. I software funzionano allo stesso modo nella forma e nella ricerca. Utilizziamo l'allineamento locale nel secondo caso perché si tratta di proteine diverse che ci aspettiamo condividano domini o motivi proteici e quindi ci aspettiamo allinei solo una parte della sequenza perché ciò che non è dominio non lo condividono ed è diverso (come ci aspettavamo, infatti, troviamo una parte centrale delle due proteine molto somigliante). Nel caso dell'allineamento locale la percentuale di identità è bassa perché le proteine sono diverse tuttavia la percentuale di similarità è più alta perché ci sono amminoacidi conservati tra le due (quando nella sequenza vediamo barre verticali tra sequenze l'amminoacido è conservato mentre quando ci sono il punto o i due punti il livello di conservazione dell'amminoacido diminuisce). Se svolgessimo su queste due seconde sequenze l'allineamento globale vedremmo peggiorare pesantemente i livelli di similarità; otteniamo livelli più bassi perché stiamo cercando di allineare due proteine molto diverse.
La percentuale di identity rispecchia il numero di elementi che hanno un match esatto tra le nostre due sequenze e si esprime in percentuale sulla lunghezza della sequenza più corta; la percentuale di similiarity, oltre a prendere questa informazione che ci dà l'identity (cioè l'informazione degli elementi che hanno un exact match) ci dà informazioni sul numero di elementi conservati o con un certo numero di conservazione (come amminoacidi con la stessa carica). I criteri con cui la conservatività è stimata dipendono dalla matrice che utilizziamo e dall'implementazione dell'algoritmo.

BLAST.

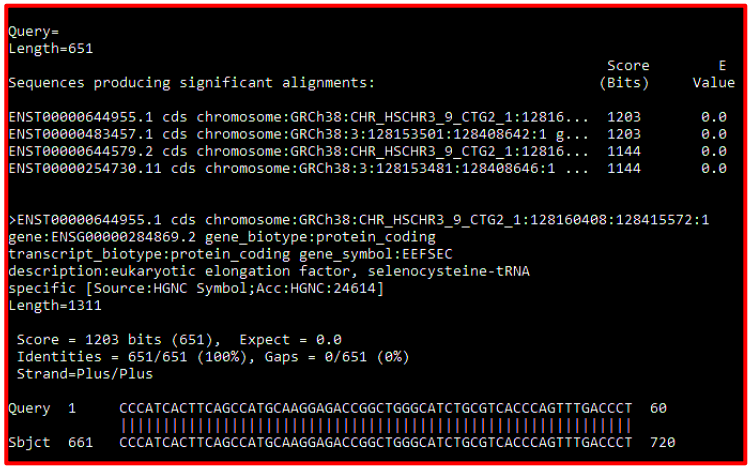
Per effettuare una ricerca di una sequenza in un database ci serve la sequenza di query (cosa vogliamo trovare), il database (la reference) e il programma (BLAST).

```
noemi@DESKTOP-TSUKVK0:~/cartella/fasta$ sudo cp ARID4A.fasta /mnt/c/Users/Noemi/Desktop
[sudo] password for noemi:
noemi@DESKTOP-TSUKVK0:~/cartella/fasta$ sudo cp /mnt/c/Users/Noemi/Desktop/tazza.jpeg ..
noemi@DESKTOP-TSUKVK0:~/cartella/fasta$ ls
ARID4A.fasta  CHD8.fasta  FOXS1.fasta  Foxs1.fasta
noemi@DESKTOP-TSUKVK0:~/cartella/fasta$ cd ..
noemi@DESKTOP-TSUKVK0:~/cartella$ ls
fasta              lncrna_genenames.txt  output.txt  text.txt
hs_annotation.gtf  mygenelist.txt        tazza.jpeg
```

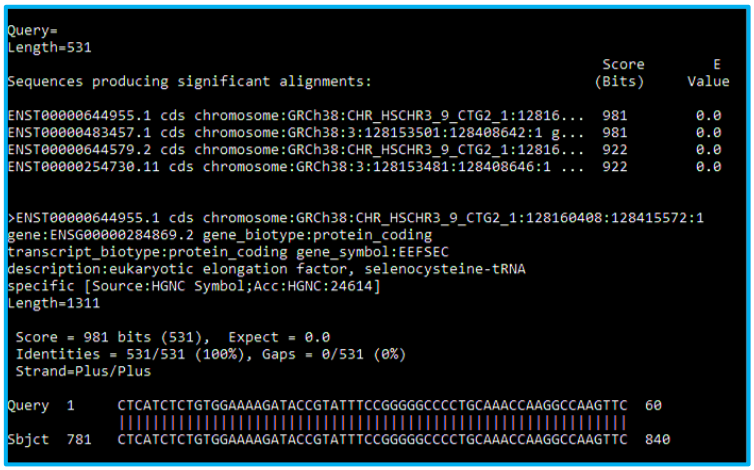
Importante: Script per spostare file da Windows a Ubuntu.

A questo punto scarichiamo BLAST dall'url fornito, estraggo, sposto sulla home di ubuntu, entro nella cartella bin e trovo numerosi file (una trentina).
Quando eseguiamo un file è importante mantenere intatto il path perché ubuntu possa eseguirlo. Il secondo passaggio è quello di creare un database per cercare la nostra sequenza; il database non è necessariamente uguale per tutti (con gli aggiornamenti a volte ci sono modifiche delle reference e gli esoni nei genomi spesso si muovono). Il nostro riferimento può essere qualunque cosa ci serva, in questo caso noi scarichiamo le sequenze codificanti. Creiamo il database su blast con makeblastdb tramite la sequenza scaricata da Ensembl poi estraiamo le ultime righe per definire una sequenza con cui fare pratica sull'allineamento. A questo punto ci ritroviamo con tutto ciò di cui abbiamo bisogno cioè il programma, il database che abbiamo costruito e la query estratto dalla reference. Possiamo quindi procedere con l'allineamento ottenendo un output che presenta il numero di hits con i loro identificativi, lo score, l'expected value e poi, uno per uno, tutti gli allineamenti delle hits. Da questo tipo di ricerca possiamo ottenere varie informazioni: in primis possiamo cercare di capire se la nostra sequenza è presente nel database; possiamo anche capire a quali altre sequenze è vicino (sequenze omologhe) e se ci sono domini conservati possiamo capirne la funzione o possiamo anche capire dove sta la nostra sequenza in un trascritto. Ora comunque abbiamo usato come sequenza seq1 (rosso che era tratta dal database quindi aveva una corrispondenza esatta; possiamo vedere, modificando la sequenza 1 con l'editor della shell (cioè nano il cui salvataggio si fa con Ctrl+O e poi si esce con CtrlX), come cambia l'output variando la sequenza. Possiamo rendere la sequenza più corta cancellando qualche riga (seq2 azzurro) oppure cancellando

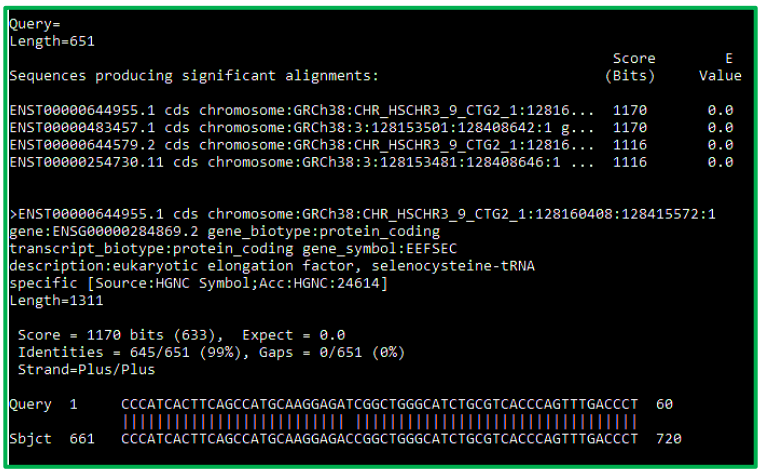
cambiando qualche nucleotide (seq3 verde) o inserendo qualche delezione (seq4 giallo).



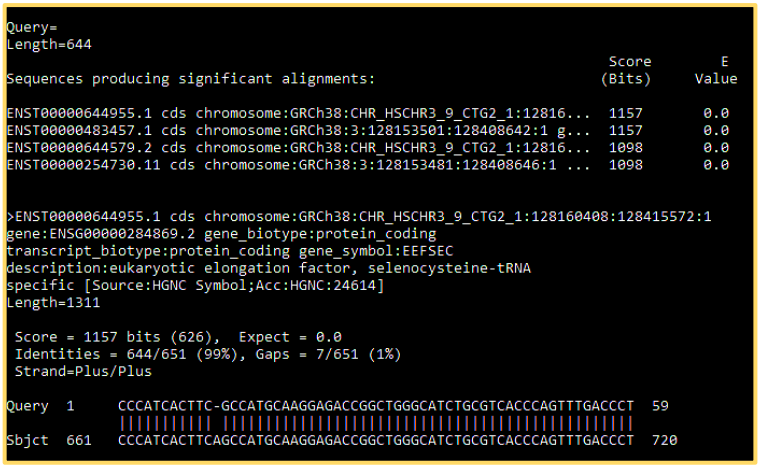
```
Query=
Length=651

                                                                      Score        E
Sequences producing significant alignments:                          (Bits)     Value

ENST00000644955.1 cds chromosome:GRCh38:CHR_HSCHR3_9_CTG2_1:12816...  1203       0.0
ENST00000483457.1 cds chromosome:GRCh38:3:128153501:128408642:1 g...  1203       0.0
ENST00000644579.2 cds chromosome:GRCh38:CHR_HSCHR3_9_CTG2_1:12816...  1144       0.0
ENST00000254730.11 cds chromosome:GRCh38:3:128153481:128408646:1 ...  1144       0.0


>ENST00000644955.1 cds chromosome:GRCh38:CHR_HSCHR3_9_CTG2_1:128160408:128415572:1
gene:ENSG00000284869.2 gene_biotype:protein_coding
transcript_biotype:protein_coding gene_symbol:EEFSEC
description:eukaryotic elongation factor, selenocysteine-tRNA
specific [Source:HGNC Symbol;Acc:HGNC:24614]
Length=1311

 Score = 1203 bits (651),  Expect = 0.0
 Identities = 651/651 (100%), Gaps = 0/651 (0%)
 Strand=Plus/Plus

Query  1     CCCATCACTTCAGCCATGCAAGGAGACCGGCTGGGCATCTGCGTCACCCAGTTTGACCCT  60
             ||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
Sbjct  661   CCCATCACTTCAGCCATGCAAGGAGACCGGCTGGGCATCTGCGTCACCCAGTTTGACCCT  720
```



```
Query=
Length=531

                                                                      Score        E
Sequences producing significant alignments:                          (Bits)     Value

ENST00000644955.1 cds chromosome:GRCh38:CHR_HSCHR3_9_CTG2_1:12816...  981        0.0
ENST00000483457.1 cds chromosome:GRCh38:3:128153501:128408642:1 g...  981        0.0
ENST00000644579.2 cds chromosome:GRCh38:CHR_HSCHR3_9_CTG2_1:12816...  922        0.0
ENST00000254730.11 cds chromosome:GRCh38:3:128153481:128408646:1 ...  922        0.0


>ENST00000644955.1 cds chromosome:GRCh38:CHR_HSCHR3_9_CTG2_1:128160408:128415572:1
gene:ENSG00000284869.2 gene_biotype:protein_coding
transcript_biotype:protein_coding gene_symbol:EEFSEC
description:eukaryotic elongation factor, selenocysteine-tRNA
specific [Source:HGNC Symbol;Acc:HGNC:24614]
Length=1311

 Score = 981 bits (531),  Expect = 0.0
 Identities = 531/531 (100%), Gaps = 0/531 (0%)
 Strand=Plus/Plus

Query  1     CTCATCTCTGTGGAAAAGATACCGTATTTCCGGGGGCCCCTGCAAACCAAGGCCAAGTTC  60
             ||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
Sbjct  781   CTCATCTCTGTGGAAAAGATACCGTATTTCCGGGGGCCCCTGCAAACCAAGGCCAAGTTC  840
```



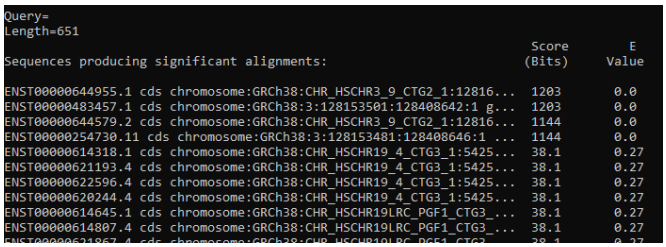
```
Query=
Length=651

                                                                      Score        E
Sequences producing significant alignments:                          (Bits)     Value

ENST00000644955.1 cds chromosome:GRCh38:CHR_HSCHR3_9_CTG2_1:12816...  1170       0.0
ENST00000483457.1 cds chromosome:GRCh38:3:128153501:128408642:1 g...  1170       0.0
ENST00000644579.2 cds chromosome:GRCh38:CHR_HSCHR3_9_CTG2_1:12816...  1116       0.0
ENST00000254730.11 cds chromosome:GRCh38:3:128153481:128408646:1 ...  1116       0.0


>ENST00000644955.1 cds chromosome:GRCh38:CHR_HSCHR3_9_CTG2_1:128160408:128415572:1
gene:ENSG00000284869.2 gene_biotype:protein_coding
transcript_biotype:protein_coding gene_symbol:EEFSEC
description:eukaryotic elongation factor, selenocysteine-tRNA
specific [Source:HGNC Symbol;Acc:HGNC:24614]
Length=1311

 Score = 1170 bits (633),  Expect = 0.0
 Identities = 645/651 (99%), Gaps = 0/651 (0%)
 Strand=Plus/Plus

Query  1     CCCATCACTTCAGCCATGCAAGGAGATCGGCTGGGCATCTGCGTCACCCAGTTTGACCCT  60
             |||||||||||||||||||||| ||||||||||||||||||||||||||||||||||||
Sbjct  661   CCCATCACTTCAGCCATGCAAGGAGACCGGCTGGGCATCTGCGTCACCCAGTTTGACCCT  720
```



```
Query=
Length=644

                                                                      Score        E
Sequences producing significant alignments:                          (Bits)     Value

ENST00000644955.1 cds chromosome:GRCh38:CHR_HSCHR3_9_CTG2_1:12816...  1157       0.0
ENST00000483457.1 cds chromosome:GRCh38:3:128153501:128408642:1 g...  1157       0.0
ENST00000644579.2 cds chromosome:GRCh38:CHR_HSCHR3_9_CTG2_1:12816...  1098       0.0
ENST00000254730.11 cds chromosome:GRCh38:3:128153481:128408646:1 ...  1098       0.0


>ENST00000644955.1 cds chromosome:GRCh38:CHR_HSCHR3_9_CTG2_1:128160408:128415572:1
gene:ENSG00000284869.2 gene_biotype:protein_coding
transcript_biotype:protein_coding gene_symbol:EEFSEC
description:eukaryotic elongation factor, selenocysteine-tRNA
specific [Source:HGNC Symbol;Acc:HGNC:24614]
Length=1311

 Score = 1157 bits (626),  Expect = 0.0
 Identities = 644/651 (99%), Gaps = 7/651 (1%)
 Strand=Plus/Plus

Query  1     CCCATCACTTC-GCCATGCAAGGAGACCGGCTGGGCATCTGCGTCACCCAGTTTGACCCT  59
             |||||||||||| ||||||||||||||||||||||||||||||||||||||||||||||
Sbjct  661   CCCATCACTTCAGCCATGCAAGGAGACCGGCTGGGCATCTGCGTCACCCAGTTTGACCCT  720
```

La seq2, più corta, rispetto a seq1 ha comunque E value molto buono ma uno score molto più basso. Seq3 è quella mutata e qui lo score è chiaramente intermedio tra seq1 e seq2 perché abbiamo introdotto mismatch e allo stesso modo funziona con la sequenza con delezioni.

Proviamo poi a modificare i parametri (ovviamente solo per il comando che forniamo e non da qui in poi) di blast e salvare i risultati: riduciamo prima la word size (lunghezza del best perfect match che vado a cercare che di default è lunga 11. In output questa volta visualizziamo molte più hits di cui sono le solite e hanno uno score molto alto mentre le altre hanno uno score molto più basso e un E-value che sale (l'E-value rappresenta la possibilità che la nostra hit sia un valore casuale o che identifichi un ritrovamento casuale); in sintesi quindi troviamo molte più hits ma molto meno affidabili. Altro parametro che possiamo richiedere è che la sequenza sia ungapped (con i gap molto penalizzato) e lo facciamo sia nella prima sequenza che nella quarta (in cui abbiamo le delezioni). Per quanto riguarda la ungapped della prima sequenza troviamo le stesse quattro hits con uno score altissimo mentre nella versione ungapped della sequenza quattro si riduce lo score e otteniamo un E-value di $5e^{-104}$ (sotto il valore di $5e^{179}$ il valore diventa 0) quindi un valore bassissimo ma presente perché la sequenza viene comunque trovata con una certa sicurezza in quelle posizioni (lo score si abbassa perché penalizziamo il gap in maniera particolare). Ovviamente questi parametri andranno adattati a quello che stiamo cercando, dobbiamo tenere in mente la nostra domanda biologica: se stiamo Blastando una sequenza di topo per vedere se ha analogo umano non ha senso essere troppo restrittivi mentre posso essere molto molto stringente se voglio identificare esattamente una sequenza che so che viene da un gene umano e



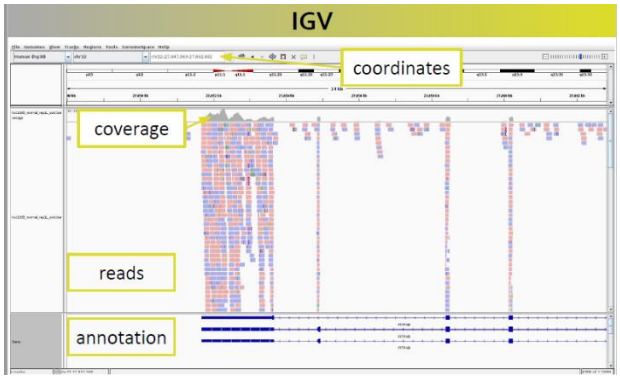
```
Query=
Length=651

                                                                      Score        E
Sequences producing significant alignments:                          (Bits)     Value

ENST00000644955.1 cds chromosome:GRCh38:CHR_HSCHR3_9_CTG2_1:12816...  1203       0.0
ENST00000483457.1 cds chromosome:GRCh38:3:128153501:128408642:1 g...  1203       0.0
ENST00000644579.2 cds chromosome:GRCh38:CHR_HSCHR3_9_CTG2_1:12816...  1144       0.0
ENST00000254730.11 cds chromosome:GRCh38:3:128153481:128408646:1 ...  1144       0.0
ENST00000614318.1 cds chromosome:GRCh38:CHR_HSCHR19_4_CTG3_1:5425...  38.1       0.27
ENST00000621193.4 cds chromosome:GRCh38:CHR_HSCHR19_4_CTG3_1:5425...  38.1       0.27
ENST00000622596.4 cds chromosome:GRCh38:CHR_HSCHR19_4_CTG3_1:5425...  38.1       0.27
ENST00000620244.4 cds chromosome:GRCh38:CHR_HSCHR19_4_CTG3_1:5425...  38.1       0.27
ENST00000614645.1 cds chromosome:GRCh38:CHR_HSCHR19LRC_PGF1_CTG3_...  38.1       0.27
ENST00000614807.4 cds chromosome:GRCh38:CHR_HSCHR19LRC_PGF1_CTG3_...  38.1       0.27
```

voglio vedere qual è l'identità del gene; altra cosa è che quando usiamo ncbi sul web vedremo che ci sono dei database di riferimento ma in locale possiamo creare database diversi ed è molto utile per organismi non troppo comuni. Possiamo anche cambiare il costo del gap e ottenere i diversi valori per le sequenze. Noi abbiamo fatto allineamenti tra nucleotidi ma ci sono programmi all'interno di Blast che permettono di fare l'allineamento proteina-proteina (protein blast). C'è la possibilità anche di blastare una sequenza nucleotidica che viene tradotta al volo contro un database di sequenze proteiche e il contrario oppure possiamo blastare nucleotidi (sia query che reference) e richiedere output proteico (utile nel caso in cui vogliamo controllare le open reading frame).

HT alignment (allineamento high-throughput)

Abbiamo bisogno per prima cosa di un reference e il nostro reference in formato FASTA si può prendere da varie piattaforme (come Ensemble o NCBI) e questa volta usiamo GENCODE, molto comoda per chi lavora in human e mouse perché permette di avere tutti i reference sia in formato FASTA che GTF o GFF3; la cosa buona è che permette, cliccando "release history" di ottenere le versioni passate dei reference perché se analizziamo dati provenienti da un paper non recentissimo possiamo prendere le annotazioni delle versioni precedenti; altra cosa comoda è che sono messi in parallelo gencode, l'ensemble e UCSC. Altra informazione utile che possiamo trovare su gencode è la specificazione dei dataformat.
Una volta che abbiamo il reference dobbiamo indicizzarlo per accelerare l'allineamento (ci fornisce la reference già indicizzata perché comunque richiede tempo. Altro file che dobbiamo fornire in input è il file FASTQ che contengono la sequenza e la qualità e hanno un formato di quattro righe in quattro righe (cioè quattro righe per ogni sequenza); la prima riga inizia con una @ e segue l'ID della sequenza e diverse informazioni, segue la sequenza, quindi una riga che inizia con + e la quarta riga è la qualità di ogni nucleotide della sequenza codificata secondo un determinato codice a seconda della macchina. Per visualizzare il file fastQ, se è normale basta un less mentre se è zippato serve uno zless. Ovviamente guardando il file capisco solo se è formattato nella maniera corretta quindi se ho le quattro righe ma non traggo informazioni sulla sequenza, nell'analisi e nel sequenziamento servirà altro.
Una volta ottenuto l'indice del reference possiamo procedere all'allineamento; il comando per l'allineamento con bowtie si esegue chiamando bowtie come "bowtie2 -x bowtie2db/referenceindicizza, --local, -1 data/sequenzar1.fastq, -2 data/sequenzar2.fastq, -S out/sequenza.sam" dove -x serve per passargli l'indice, -1 e -2 indicano i due orientamenti diversi della stessa sequenza (quando sequenziamo paired end abbiamo due reads che si specchiano e hanno orientamento opposto e vengono, però, dallo stesso frammento di DNA quindi sappiamo che la prima read del file è ad una definita distanza dalla prima read del secondo file quindi l'allineatore riconosce questa cosa e allinea le reads accoppiate e per dirci qual è l'una e qual è l'altra le nominiamo con -1 e -2) e -S l'output della nostra analisi (si chiama -S perché viene scritto nel formato del file di allineamento che si chiama SAM). In bowtie il parametro "local" ci permette di fare l'allineamento con clipping, cioè permette all'allineatore di rompere la read (fare clipping della read) se non allinea direttamente sulla reference. Questo passaggio è molto comodo perché stiamo allineando read che vengono da trascritti ottenuti tramite esperimenti di RNA seq e li stiamo allineando con un reference genomico quindi avremo read che vengono da due esoni, uno dietro l'altro, mentre nel reference ci sarà in mezzo l'introne; permettendo all'allineatore di spezzare la read riusciamo comunque ad allineare sulla reference nonostante lì ci sia l'introne (idealmente, infatti, in un esperimento di RNA seq vorremmo trovarci a lavorare con il trascrittoma e invece stiamo usando per comodità un genoma che è già indicizzato- mentre l'indicizzazione avrebbe richiesto molto tempo- e questo richiede di utilizzare alcune altre potenzialità con piccoli accorgimenti).
L'output dell'allineamento è un file di formato SAM (Sequence Alignment/Map format) che è un formato di file di testo e contiene l'informazione dell'allineamento, cioè da dove a dove allinea il file in questione e anche tutta una serie di ulteriori informazioni sulla qualità dell'allineamento e su come l'allineamento viene (se è perfetto, se ci sono mismatch). I file SAM che derivano dall'allineamento sono file molto grandi per cui di solito si comprimono in formato BAM (BGZF compressed SAM format) con una compressione che

permette l'indicizzazione e il recupero della reads; se proviamo ad aprire un file BAM con less ci rendiamo subito conto della compressione perché è in binario. Arriva a ridurre la dimensione di molto quindi una volta ottenuta l'informazione possiamo salvare direttamente l'allineamento in BAM e liberare spazio nel disco. Una cosa utile del BAM è che può avere gli allineamenti ordinati per nome della read o per coordinata o non ordinati (nel caso siano ordinati per coordinata il BaAM può essere accompagnato da un file .bai che è un file indice che produciamo per ogni BAM e permette, ad alcuni programmi che lo richiedono (che usiamo successivamente nella pipeline) di accedere al BAM in maniera facilitata e quindi ci permette di andare a pescare direttamente gli allineamenti sulle coordinate che ci servono.
I passaggi successivi nella pipeline all'allineamento con Bowtie sono proprio questi: compressione del file SAM in BAM (samtools view -bS out/hcc1395_normal_rep1.sam, out/hcc1395_normal_rep1.bam). Il comando view del tool di sam ci fa scrivere -bS dove b significa che l'output è in BAM ed S che l'input è in SAM. I samtools sono una collezione di comandi che ci permettono di fare molte cose e sono sempre consultabili così come tutte le opzioni dei comandi.
Il secondo passaggio, una volta ottenuto il BAM, è di ordinarlo secondo le coordinate e quindi usiamo il comando sort che prenderà in input il BAM appena creato e darà in output lo stesso file ma ordinato.
Il terzo passaggio richiede di creare l'indice bai e per far questo possiamo semplicemente dare in input il BAM ordinato per coordinata e l'output viene creato automaticamente da un altro samtool ed è un ".bam.bai". Il formato bai è un file che accompagna il bam, non ha senso di esistere da solo, e contiene un indice del file di allineamento che è richiesto da alcuni programmi che usiamo successivamente (come ad IGV) per accedere direttamente alle coordinate utili. Successivamente uno dei passaggi che si possono fare per valutare a livello visivo il dato è di caricare gli allineamenti su un visualizzatore, come IGV.
IGV prima di tutto è comodo perché lo abbiamo in locale e possiamo caricarci i nostri dati (se vogliamo vedere i dati di un paper e analizzarli, questi sono disponibili online possiamo scaricarli e verificarli tramite il software in locale. IGV è un software molto semplice, lo possiamo aprire dalla linea di comando semplicemente digitandone il nome (dopo averlo installato) e questo viene eseguito e si apre nell'interfaccia grafica [per chi ha windows invece non deve avviarlo dalla linea di comando ma dai programmi da avviare, l'unica cosa un po' più delicata è che dobbiamo fare il passaggio dei file bam e bai dall'ambiente dell'interfaccia grafica di ubuntu sul desktop di windows]. IGV è intuitivo, selezioniamo un reference dal menu a tendina dei reference (in cui possiamo anche caricare roba) e da "file" selezioniamo il nostro bam (che porterà con sé, senza selezionarlo, il bai) e si formeranno due track nella parte superiore che riporteranno tutti i cromosomi quindi noi potremo zoomare fino a che non vediamo le coordinate. Nella prima riga troviamo il coverage che ci presenta una curva tanto più alta quanto più coverage abbiamo nella parte inferiore sono rappresentate le reads (rosso e blu per distinguere forward e reverse); dove abbiamo più alto il picco di coverage, abbiamo anche più sequenze.
Nella parte inferiore vediamo introni ed esoni e se usiamo la funzione expanded (che troviamo con il tasto destro) ci menziona anche le varie isoforme di un trascritto.



Lo pseudomapping invece avviene in due passaggi. Prima si prepara l'indice del reference quindi si allinea e si ottengono dei BAM (ma la quantificazione va fatto in maniera separata). Con l'approccio di Salmon e di Callisto possiamo evitare lo step di allineamento e andare direttamente in un unico step alla quantificazione. Per la quantificazione Salmon vuole avere informazioni riguardo il dataset (-l vuole informazioni sulla libreria per sapere come è costruito l'allineamento quindi paired end, single end, l'orientamento delle reads, se conosciamo lo strand ecc) e se non le abbiamo chiediamo a Salmon di cercarsele da solo (A).

1. Indexing
   - salmon index -t gencode.v33.transcripts.fa.gz -i gencode.v33.transcripts.index
2. Quantification
   - salmon quant -i gencode.v33.transcripts.index -l A -1 data/hcc1395_normal_rep1_r1.fastq -2 data/hcc1395_normal_rep1_r2.fastq -o transcript_quant

Quando abbiamo l'index pronto e l'input pronto salmon ci mette pochi secondi a fornirci la quantificazione dei trascritti (mentre il mapping con gli stessi file ci ha messo 10 minuti). Ci sono vari file di output costruite: una fondamentale e alcune informazioni aggiuntive correlate. L'output che ci interessa è quant e presenta 5 colonne: nome del trascritto, lunghezza del trascritto, la lunghezza effettiva, la quantificazione dell'espressione in TPM (Transcript Per Million) e la stima del numero di reads. Quindi questo metodo è più rapido sia a livelli di tempo che di domande da fare perché in due passaggi fornisce la quantificazione e possiamo già procedere, ad esempio, a esperimenti di espressione differenziale. L'allineamente è comunque valido per una serie di altri tipi di esperimento (se voglio allineare sul genomico perché ho sequenziato un clone diverso da quello di riferimento è utile l'allineamento o anche se voglio indagare regioni specifiche del genoma oppure trovare transcritti che credo non siano annotati ma siano presenti nel mio campione o tessuto.

- **Name** — This is the name of the target transcript provided in the input transcript database (FASTA file).
- **Length** — This is the length of the target transcript in nucleotides.
- **EffectiveLength** — This is the computed effective length of the target transcript. It takes into account all factors being modeled that will effect the probability of sampling fragments from this transcript, including the fragment length distribution and sequence-specific and gc-fragment bias (if they are being modeled).
- **TPM** — This is salmon's estimate of the relative abundance of this transcript in units of Transcripts Per Million (TPM). TPM is the recommended relative abundance measure to use for downstream analysis.
- **NumReads** — This is salmon's estimate of the number of reads mapping to each transcript that was quantified. It is an "estimate" insofar as it is the expected number of reads that have originated from each transcript given the structure of the uniquely mapping and multi-mapping reads and the relative abundance estimates for each transcript.

[Aggiunte sulla lezione appena fatta. Abbiamo usato BOWTIE per allineare le nostre sequenze su reference genomico; l'utilizzo del reference genomico era per l'esercitazione perché avevamo la possibilità di scaricarlo già indicizzato mentre produrre l'indice per il trascrittoma ci avrebbe richiesto molto tempo e siccome abbiamo usato il reference genomico abbiamo usato l'opzione di BOWTIE local per spezzettare la nostra reads consentendo un allineamento più elastico; in generale però si usa il trascrittoma.
Inoltre abbiamo allineato, sempre con BOWTIE, due FASTQ files perché i nostri dati da esempio venivano da una library paired end; quando si costruisce una library dal cDNA si può sequenziare il frammento da una parte o da tutte e due e quindi possiamo avere librerie paired end o single end; la libreria single end ci da un solo file fastQ mentre la paired ce ne da due per campione (in uno avremo le reads forward e nell'altra le reverse). In particolare le librerie paired end possono essere più utili delle single end quando si fa ad esempio assemblaggio di trascritti oppure quando vogliamo identificare isoforme diverse. Anche il tipo di library comunque non è comandato ma si sceglie quando si prepara il setting per l'esperimento.
Altro appunto sono i file di appunto delle reference. La reference viene indicizzata per permettere un accesso veloce e randomizzata alla reference; BOWTIE per fare questo necessita di 6 file diversi che genera in autonomia e il fatto che ci siano questi file diversi è una questione di indicizzazione del file, problema informatico di come il reference viene indicizzato; noi per lanciare l'allineamento usiamo il base name che è quella comune a tutti quanti che finisce per "index".
Ultima cosa sono le opzioni dei programmi. Per i programmi della linea di comando del terminale sappiamo che possiamo utilizzare "man comando" o "comando -h" o "comando –help"; quando si tratta di programmi che abbiamo installato noi, e quindi non sono già presenti nel nostro UBUNTU di solito funziona il –help ma nel dubbio proviamo a lanciare il comando senza opzioni o input e ci indirizza all'help. In alternativa online per ogni programma ci sono pagine web con il manuale completo.]
Allineamento multiplo. Per effettuare un allineamento multiplo sfruttiamo un tool online dell'ebi che si chiama Clustal Omega in cui, come avevamo fatto con gli allineamenti globale e locale, diamo in input al software online le nostre sequenze proteiche (usiamo un file Fasta che contiene le sequenze amminoacidiche di cinque proteine che vogliamo confrontare). Scorrendo un po' tra i risultati riusciamo a visualizzare che ci sono alcune porzioni in cui le cinque sequenze allineano e quindi visualizziamo i domini. Ci sono in verità varie aree di questo software, oltre alla regione che ci permette di fare allineamento multiplo c'è una sezione dedicata alla costruzione di alberi filogenetici, cladogrammi o alberi reali (che rendono evidente quanto alcune di queste proteine siano assolutamente vicine tra loro).
Quando facciamo l'allineamento multiplo con sequenze nucleotidiche (nel nostro file in input le sequenze sono molto più corte e molte di più); per queste sequenze qui semplicemente salviamo il nostro alignment file così com'è da usare in seguito per analisi secondarie. L'allineamento multiplo ha vari output ma si vede quasi sempre allo stesso modo, cambiano alcuni dettagli.

Questo è un esempio di multiple allignment che possiamo fare quando ci serve comparare il nostro gene di interesse in varie specie, ad esempio, oppure come nel caso visto ora quando vogliamo confrontare proteine che appartengono alla stessa famiglia e sono leggermente diverse.
JASPAR database è un database per siti di legame su DNA per fattori di trascrizione (transcription factor binding site), proteine di varie specie. Sono dati validati sperimentalmente e sono correlati anche a matrici. Questo sito ci permette di vedere se ci sono particolari motivi che una proteina utilizza per legarsi. Per farlo diamo in pasto a biopython un file fasta contenente i Pax2 sites, siti di legame del fattore Pax2 in mouse (sono sequenze che provengono da regioni diverse su cui pax2si lega). Importiamo la libreria Biopython motifs e carichiamo il file pax2.sites; quindi facciamo un'indentazione e diamo il nome pax2 alle istanze che andiamo a trovare.
Abbiamo quindi visto che una proprietà del nostro oggetto che si chiama instances ("Pax2.instances") sono le sequenze che abbiamo caricato. Un'altra proprietà dell'oggetto Pax2 si chiama counts ("Pax.counts") che fa il conto dei nucleotidi in ogni posizione; abbiamo il nostro alfabeto formato dai 4 nucleotidi e, in questo caso, il numero di posizioni che va da 1 a 7 (numero di posizioni della nostra sequenza di binding). Questa proprietà ci costruisce la matrice di frequenza stampandoci per ogni posizione quante volte osserviamo ogni nucleotide. Possiamo accedere ad ogni posizione per farci restituire delle informazioni riguardo i nucleotidi.

```
>>> Pax2
<Bio.motifs.jaspar.Motif object at 0x7f04b76d5eb8>
>>> print(Pax2.counts)
        0      1      2      3      4      5      6      7
A:  10.00   7.00   3.00   0.00  26.00   2.00   2.00   1.00
C:   7.00   1.00   2.00  28.00   1.00  17.00   0.00  11.00
G:   5.00  21.00   0.00   1.00   3.00   1.00  19.00  11.00
T:   9.00   2.00  26.00   2.00   1.00  11.00  10.00   8.00

>>> Pax2.counts['A']
[10, 7, 3, 0, 26, 2, 2, 1]
>>> Pax2.counts[:,4]
{'A': 26, 'C': 1, 'G': 3, 'T': 1}
>>> Pax2.counts['A',4]
26
```

Counts poi ha una proprietà che si chiama consenso (print {Pax2.counts.consensus}, AGTCACGC) che ci fornisce la sequenza consenso per il nostro insieme di sequenze ossia la sequenza più probabile, i cui nucleotidi si verificano più frequentemente. Possiamo in maniera simile (print {Pax2.counts.degenerate_consensus}, NGTCAYKN) ricavare il consenso degenerato che, utilizzando il codice IUPAC ci dà il consenso degenerato rispetto alla nostra sequenza.
Proseguendo, noi vogliamo costruire una matrice di probabilità e per fare questo usiamo la funzione "normalize" che si applica a "count"; come abbiamo visto normalizziamo i calcoli della matrice dividendo per il numero di istanze nell'allineamento cioè divide la matrice per il numero di sequenze, ottenendo quindi la probabilità di ogni nucleotide in ogni posizione (ppm= Pax2.counts.normalize()).

```
>>> ppm = Pax2.counts.normalize()
>>> print(ppm)
        0      1      2      3      4      5      6      7
A:  0.32   0.23   0.10   0.00   0.84   0.06   0.06   0.03
C:  0.23   0.03   0.06   0.90   0.03   0.55   0.00   0.35
G:  0.16   0.68   0.00   0.03   0.10   0.03   0.61   0.35
T:  0.29   0.06   0.84   0.06   0.03   0.35   0.32   0.26

>>> pwm = ppm.log_odds()
>>> print(pwm)
        0      1      2      3      4      5      6      7
A:  0.37  -0.15  -1.37   -inf   1.75  -1.95  -1.95  -2.95
C: -0.15  -2.95  -1.95   1.85  -2.95   1.13   -inf   0.51
G: -0.63   1.44   -inf  -2.95  -1.37  -2.95   1.29   0.51
T:  0.22  -1.95   1.75  -1.95  -2.95   0.51   0.37   0.05
```

Dalla matrice di probabilità vogliamo poi ottenere la matrice di peso e lo facciamo applicando alla matrice ppm il log odds (pwm=ppm.log_odds()). In questo caso utilizziamo un background di default che vuol dire che ogni nucleotide sta a 0.25. Se, infine, il nostro organismo è particolare e non ha un background per cui ogni nucleotide è espresso per il 25% ma ha ad esempio un 40% di GC content possiamo settare un background (background= {"A":0.3, "C":0.2, "G":0.2, "T":0.3}) e costruire una nuova matrice che sarà una pwm2 che calcoliamo sempre con il log_odds ma dando in input il background da noi definito (pwm2= ppm.log_odds (background)).
Un'altra possibilità che ci dà biopython è quella di rappresentare graficamente un web logo che rappresenti la quantità di informazione che ottengo dalla scoperta di un nucleotide. Costruirlo è molto semplice perché esiste un tool online (weblogo.threeplusone.com) che si chiama weblogo che ci permette di creare loghi facilmente solo caricando il nostro file ottenuto da Clustal, cambiando i parametri per definire più che altro la grafica del logo ma anche se ignorare i casi più bassi.

Filogenetica

Iniziamo chiedendoci quale sia il rapporto tra la sequenza amminoacidica di Chd1 in specie diverse. Valutiamo 9 proteine, quindi 9 file fasta della famiglia Chd e quindi 9 sequenze proteiche della stessa proteina in diversi organismi. Per pensare di creare un albero con questi dati dobbiamo cominciare facendo un allineamento multiplo (perché sono sequenze non allineate e separate tra loro). Dai singoli file fasta ognuno contenente una sequenza proteica della famiglia di CHD (CHD1/9) in umano dobbiamo generare un unico file che contenga tutte le sequenze fasta (dobbiamo farne un merge): usiamo il comando cat e un'espressione regolare per indicare tutti i file nella cartella che cominciano per CHD e sono seguiti da un numero (cat CHD*.fasta > humanCHD.fasta). Dobbiamo poi fare un secondo merge per creare un file che contenga tutte quante le sequenze di CHD1 nei diversi organismi e lo facciamo allo stesso modo (cat CHD1.fasta CHD1_*.fasta>Chd1_organisms.fasta). Ora abbiamo i nostri due file e possiamo procedere facendo l'allineamento multiplo tramite Clustal Omega come abbiamo già visto (ovviamente bisogna dare in input separatamente prima la famiglia di CHD e poi i CHD1 nei diversi organismi) ma questa volta chiediamo di ricevere in output un fasta. Ora che abbiamo le nostre sequenze allineate torniamo alla sequenza iniziale cioè in che rapporto stanno le sequenze CHD di tutti questi organismi quindi cerchiamo di costruire un albero con un approccio di maximum likelihood utilizzando i dati allineati e un modello evolutivo. L'approccio di maximum likelihood permette di creare l'albero dalla probabilità più alta dato il modello evolutivo e per fare questo utilizziamo un programma chiamato RAxML (Randomized Axelerated Maximum Likelihood), programma di massima probabilità basato sull'inferenza di grandi alberi filogenetici. Abbiamo fatto una prima prova con raxml utilizzando il modello BLOSUM62.

Il comando è: raxm1HPC -s Chd1_organisms_aln.fasta -m PROTGAMMABLOSUM62 -n Chd1_organisms_blosum62 -p 58436851.

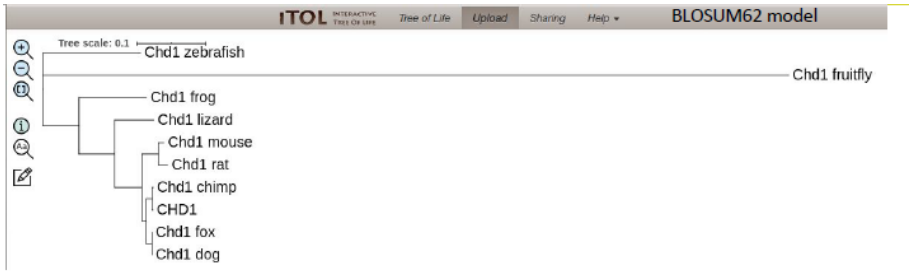
raxml1HPC è il comando per avviare il programma, -S introduce il file di sequenze allineate, -m introduce il modello (PROTGAMMABLOSUM62 è un modello per proteine che spiega una disposizione delle mutazioni che è γ, molto usata perché facile da calcolare a livello matematico e non c'è una definita ragione biologica ma è la più usata soprattutto per dataset piccoli e il modello, nel dettaglio come visto a lezione, è blosum62), -n introduce il nome dell'output che vogliamo creare e -p vuole un random number seed cioè dobbiamo dargli un numero casuale che (serve per tutti gli algoritmi che richiedono randomizzazione) nel caso specifico serve per l'inizializzazione dell'albero prima che avvenga l'ottimizzazione di maximum likelihood (diamo noi il numero casuale invece di farlo fornire dall'algoritmo per avere la possibilità di ripetere l'algoritmo essendo certi di ottenere lo stesso output). Il nostro output sarà costituito da diversi file e quello che ci interessa in maniera particolare è il bestTree perché è il nostro risultato: questo file rappresenta la struttura di un albero mettendo in parentesi gli ID delle nostre sequenze e la lunghezza dei rami; l'output in questo file però è soltanto una striscia di numeri e sequenze quindi dobbiamo visualizzarlo in altra maniera.

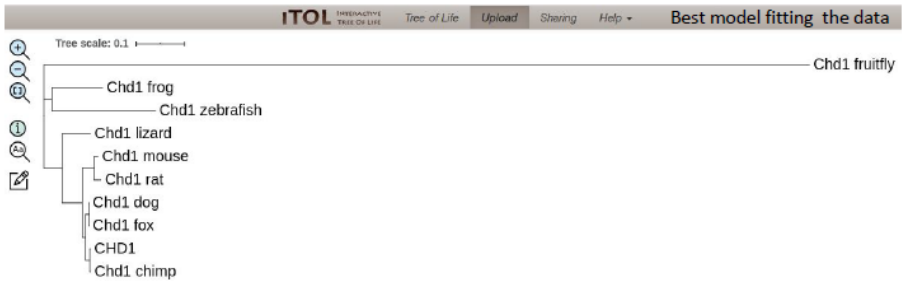
Possiamo anche confrontare l'output venuto fuori quando siamo noi a fornire un modello con quello venuto fuori con un modello scelto da RAxML modificando, nella riga di comando precedente "-m PROTGAMMABLOSUM62" con "PROTGAMMAAUTO"; in questo modo il problema è un po' più semplice perché non dobbiamo pensare a qual è il modello più appropriato per rispondere alla nostra domanda biologica e per le sequenze che stiamo guardando ma lo fa RAxML per noi. Questo metodo è un po' più lento rispetto al precedente perché deve testare tanti modelli.

Abbiamo scelto RAxML perché ci permette di provare a fare un albero con maximum likelihood ma ci sono moltissimi comandi all'interno di RAxML e le prove possono essere tantissime. Altro metodo interessante ma che non riusciamo a testare è quello del Bootstrapping perché in questo modo avremmo potuto dare un valore di significatività ai nostri alberi. Anche con questa seconda metodologia l'output che ci interessa è il bestTree che restituisce delle righe in cui si susseguono l'ID e la lunghezza dei rami ma ad occhio sappiamo dire che il file rappresenta un albero ma non sappiamo distinguere a occhio, leggendo, le differenze tra i due.

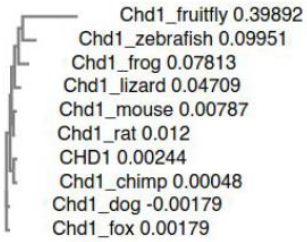
Dobbiamo quindi visualizzare in maniera grafica i risultati e lo facciamo tramite iTOL (interactive Tree Of

Life); dal momento che abbiamo i due alberi (uno costruito con modello BLOSUM e l'altro con modello scelto da RAxML) apriamo due pagine di iTOL per fare i due upload e sarà lui stesso a costruire graficamente i collegamenti sulla base dei nostri dati.

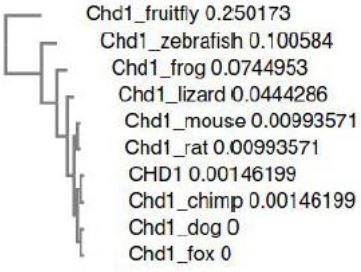




Diventa presto evidente che il modello conta perché gli alberi risultano piuttosto diversi non solo nelle distanze ma anche nelle connessioni. In entrambe nessuno dei nostri organismi fa da radice agli altri.
Tra organismi molto vicini come umano e chimpanzee oppure il cane e la volpe o topo e ratto le differenze nei due alberi quasi non si vedono mentre la differenza è molto più evidente mano a mano che guardiamo organismi più distanti. A far cambiare l'albero è il cambiamento, tra un modello e l'altro, di probabilità di visualizzare una stessa mutazione tra un organismo e l'altro. I modelli, a seconda del modo in cui sono costruiti, danno un peso diverso a una mutazione di un tipo rispetto a una mutazione di un altro tipo; confrontando le stesse sequenze e costruendo lo stesso albero ma con modello diverso cambia la distanza tra organismi. Quindi la possibilità di chiedere a RAxML di trovare il best model fitting the data è molto utile perché altrimenti manualmente dovremmo costruire gli alberi e confrontarli e scegliere l'albero che ci è utile. La distanza in verticale è puramente grafica mentre la distanza in termini di tempo è quella tra l'organismo e il nodo in orizzontale (la distanza con un altro organismo si fa andando a misurare i rami da un organismo all'altro).
Abbiamo quindi visto il metodo di maximum likelihood per costruire un albero con due modelli diversi e ora possiamo provare a costruire l'albero utilizzando le stesse sequenze di CHD1 nei vari organismi con due metodi diversi, entrambi basati sulla distanza e sono il neighbour joining e l'UMPGA (Unweighted Pair Group Method with Arithmetic mean). Per provare questi due approcci utilizziamo un tool online dell'enbi che si chiama Simple Phylogeny (con grafica e interfaccia molto simile agli altri software dell'enbi quindi Clustal, Nedeelman e Water); l'input si da allo stesso modo, carichiamo il nostro file di allineamento multiplo e selezioniamo il metodo di clustering neighbour joining. L'output a livello di testo è fatto come quelli di prima e lo possiamo salvare; lo guardiamo in versione reale perché nel cladogramma la distanza tra i rami è sempre la stessa indipendentemente dai nostri dati (tra reale e cladogramma non cambia la struttura ma cambia di molto la lunghezza dei rami perché in real sta in relazione con la distanza tra organismi).

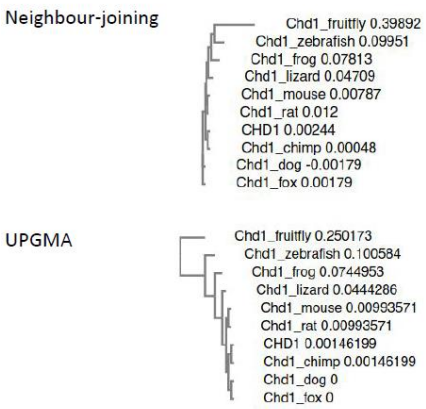


Sempre sullo stesso sito online possiamo costruire l'albero tramite UPGMA: carichiamo lo stesso file in input ma questa volta usiamo come metodo UPGMA e sottomettiamo (e salviamo allo stesso modo l'output). Anche questa volta otteniamo in output il file con gli ID delle sequenze e le distanze tra rami. Questo software online comunque è molto più rapido di RAxML perché prima di tutto visualizziamo subito l'output delle sequenze come albero grafico e testuale quindi questo secondo metodo è molto più diretto. Inoltre qui possiamo modificare o customizzare la struttura grafica dell'albero in maniera ridotta: mentre in RAxML abbiamo tante possibilità di gestire la struttura dell'albero e soprattutto RAxML ci permette di fare Bootstrapping che qui non possiamo fare; qui restituisce un albero di cui però non possiamo testare la significatività e non possiamo avere una misura. Questo secondo quindi è un tool



che viene facile da usare in modalità da remoto ma a seconda dell'analisi che vogliamo fare potrebbe essere una buona idea che ci consenta di fare Bootstrapping per ottenere un supporto statistico per il nostro albero.

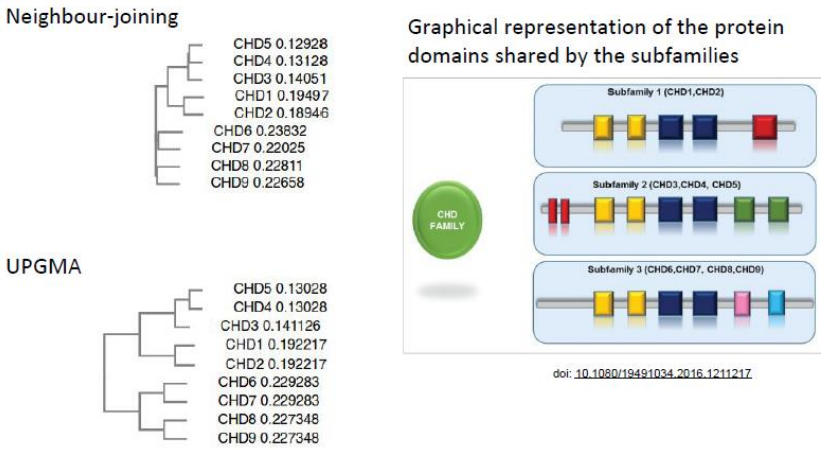
Vogliamo quindi confrontare i due alberi costruiti con NJ e UPGMA. Il metodo UPGMA prese due sequenze trova il common ancestor mettendolo a metà della distanza tra i due e parte dalle foglie (per esempio tra volpe e cane il common ancestor è posto a metà fra i due e così uomo e scimpanzee e topo e ratto ma anche lucertola e tutto il cluster sottostante e l'ancestor sta a metà) mentre il Neighbour joining utilizza il minimum evolution principle cioè vogliamo trovare un albero che abbia la minima distanza totale tra alberi quindi minimizza le distanze tra coppie di sequenze infatti i rami sono molto corti e il più lungo è quello del moscerino della frutta perché ha la distanza maggiore rispetto agli altri organismi.



Teniamo presente che questi due metori sono entrambi basati sulla distanza e che qui stiamo guardando alberi creati in base alle sequenze di CHD1: dati gli organismi che abbiamo scelto è molto probabile che scegliendo un'altra proteina l'albero sia molto simile perché in linea evolutiva è ciò che ci aspettiamo in linea di massima indipendentemente dalla proteina ma è fondamentale ricordare che è costruito sulle sequenze della proteina.

Possiamo quindi caricare gli output in formato file di testo ottenute sia per nj che per upgma (ottenuti con simple phylogeny) su iTOL come avevamo fatto per RAxML. Le strutture rimangono analoghe a quanto visto precedentemente ma è graficamente più evidente ogni distanza. Qui possiamo ancora cambiare la struttura dell'albero ma comunque cambia la rappresentazione ma il senso dell'albero rimarrà il medesimo; possiamo costruire un albero unrooted con forma diversa ma sarà mantenuta la vicinanza tra alcuni organismi e la distanza con altri.

Avevamo poi allineato le sequenze dalle proteine CHD di uomo e possiamo costruire l'albero sia in metodo neighbour joining e UPGMA e confrontarli.



In questo caso l'output è più semplice da analizzare: i rapporti delle proteine rimangono nello stesso ordine nei due metodi e anche la distanza è mantenuta. Questa distanza si rispecchia poi analizzando i domini delle proteine che infatti sono più simili nelle proteine più vicine.

Ultima cosa è che per avere supporto significativo dell'albero dovremmo sempre fare un'analisi di bootstrapping o altre che diano una valutazione statistica.

Learning from data

Andiamo a scoprire gli approcci di machine learning: iniziamo a usare un dataset che viene da un esperimento di RNA seq e che ci dà la possibilità di esplorare dati reali. Partamo dalla quantificazione e andremo avanti cercando di vedere passo passo i passaggi che fanno parte dell'analisi di RNAseq e allo stesso tempo proviamo gli algoritmi visti nella teoria. Il dataset che usiamo si chiama airway ed è contenuto in un pacchetto R e si tratta di quattro linee cellulari delle vie aeree che sono state trattate con dexametasone quindi il nostro dataset ha un totale di 8 campioni. Nell'esercitazione sull'highthroughput abbiamo visto diversi algoritmi di allineamento delle reads che vengono da un esperimento di sequencing e abbiamo visto anche un approccio di quantificazione di trascritti che salta lo step dell'allineamento che era

la quantificazione tramite Salmon. Supponiamo che invece del pacchetto che abbiamo scaricato tramite R noi avessimo i dati grezzi in formato FastQ quindi potremmo quantificarli con Salmon utilizzando il comando già visto e questo viene fatto per tutti gli 8 campioni e ci restituisce un output nella forma di un file che si chiama quant.sf che contiene delle informazioni che sono nome del trascritto, la lunghezza (numero di nucleotidi), la lunghezza effettiva (misurata tenendo conto di quanto allinea e di quanto è usato il trascritto), una quantificazione dell'abbondanza del trascritto in TPM e il numero delle reads stimate che allineo sul trascritto. Per i passaggi successivi quindi dovremo prendere queste informazioni da ciascuno dei file degli otto campioni e proseguo con la pipeline.
La nostra situazione è leggermente diversa perché per comodità utilizziamo i dati airway che contengono già tutte le informazioni in formato tabulare. Per caricare tutti questi dati in R se avessimo avuto file FastQ di partenza e li avessimo quantificati con Salmon avremmo dovuto eseguire alcuni passaggi aggiuntivi per tabulare i dati ottenuti nei singoli files.
Considerando la situazione a partire dai dati airway possiamo lanciare la library ("airway") su Rstudi; una volta caricata, siccome questi dati non li abbiamo presenti fisicamente in locale ma fanno parte del pacchetto dobbiamo creare un path, come il path di una directory, per andare a recuperare questi dati (per farlo forniamo il comando dir <- system.file ("extdata", package="airway", mustWork=TRUE). Quindi vediamo cosa c'è nella cartella richiamando list.files(dir) che mostra dati di cui non sappiamo molto: abbiamo un file gtf con un'annotazione umana, una sample table che rappresenta gli output di Salmon, dei file bam in subset e una sample table che contiene informazioni sui campioni e un file di testo che contiene informazioni sull'origine dei dati.
A questo punto carichiamo le informazioni dei nostri campioni in un dataframe che chiamiamo colldata ma prima dobbiamo creare un path dove la funzione read.csv può andare a cercare la nostra tabella. Quando leggiamo la tabella dobbiamo dare il path e il nome del file, csv.file, e diciamo che i nomi delle read stanno nella prima colonna e che non vogliamo che le stringhe vengano convertite a factors. La tabella che abbiamo costruito ci fornisce diverse informazioni: il nome dei campioni (che hanno un identificativo unico), il nome della RUN che è poi anche il nome delle righe per noi, experiment sample e BioSample che hanno identificativi separati che si rifanno al database di partenza per cui non è che si caricano i dati e si da un nome ma bisogna definire chiaramente ogni campione come è stato prodotto, con che protocollo è stato prodotto, con cosa è strato trattato, quanti passaggi ha subito la linea cellulare, che tipo di linea cellulare è, da dove viene, protocollo di crescita e altre informazioni molto dettagliate.

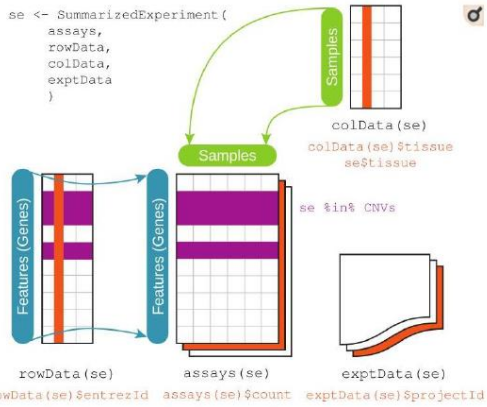
Load the sample information file:

```
csvfile <- file.path(dir, "sample_table.csv")
coldata <- read.csv(csvfile,row.names=1,stringsAsFactors=FALSE)
coldata
```

```
> coldata <- read.csv(csvfile, row.names=1, stringsAsFac
ors=FALSE)
> coldata
           SampleName    cell   dex albut        Run
SRR1039508 GSM1275862  N61311 untrt untrt SRR1039508
SRR1039509 GSM1275863  N61311   trt untrt SRR1039509
SRR1039512 GSM1275866 N052611 untrt untrt SRR1039512
SRR1039513 GSM1275867 N052611   trt untrt SRR1039513
SRR1039516 GSM1275870 N080611 untrt untrt SRR1039516
SRR1039517 GSM1275871 N080611   trt untrt SRR1039517
SRR1039520 GSM1275874 N061011 untrt untrt SRR1039520
SRR1039521 GSM1275875 N061011   trt untrt SRR1039521
           avgLength Experiment    Sample
SRR1039508       126  SRX384345 SRS508568
SRR1039509       126  SRX384346 SRS508567
SRR1039512       126  SRX384349 SRS508571
SRR1039513        87  SRX384350 SRS508572
SRR1039516       120  SRX384353 SRS508575
SRR1039517       126  SRX384354 SRS508576
SRR1039520       101  SRX384357 SRS508579
SRR1039521        98  SRX384358 SRS508580
           BioSample
SRR1039508 SAMN02422669
SRR1039509 SAMN02422675
SRR1039512 SAMN02422678
SRR1039513 SAMN02422670
SRR1039516 SAMN02422682
SRR1039517 SAMN02422673
SRR1039520 SAMN02422683
SRR1039521 SAMN02422677
> coldata<- coldata[1:]
```

Come abbiamo detto quando abbiamo quantificazioni ottenute con Salmon possiamo volerle importare in R e per farlo creiamo una colonna nella nostra tabella coldata che si chiama names e a cui corrispondono i dati che stavano nella colonna RUN; costruiamo il PATH con come directorty la cartella quants, la colonna names che contiene i nostri campioni e il nome del file Salmon. Importata la quantificazione otteniamo una tabellina che riporta la quantificazione dei trascritti e il numero dei campioni e questo è il punto di partenza quando lavoriamo con dati di quantificazione dei trascritti.
Tornando al nostro RNAseq su 4 campioni in due condizioni (quindi 8 campioni), possiamo vedere come è fatto il summarized object. Il summarized object è un oggetto di R formato da tre parti: la prima che si chiama coldata è molto simile, se non uguale, alla tabella di cui eravamo in possesso (una tabella che contiene i nomi dei campioni); una seconda parte chiamata rowData che contiene le features che sono nel nostro caso i geni o trascritti; poi abbiamo la tabella importante centrale che è quella che contiene i nostri valori di espressione e si chiama assays; L'assays contiene i valori di espressione per tutti i campioni nelle

colonne e per tutti i geni nelle righe. L'oggetto, se, lo abbiamo visto prima ed è la tabella con trascritti e campioni; per accedere a queste tre tabelle separatamente si usa la sintassi parte(esperimento) quindi ad esempio colData(se) o rowData(se).



Quindi abbiamo caricato il dataset intero che contiene tutti e 8 i campioni utilizzando la funzione data. In questo caso l'oggetto si chiama gse (ranged summarized experiment). Ranged perché la parte dei row data sono range genomici. Nelle righe sono contenute tre informazioni: counts, abundance e lenghts ciopiè esattamente l'output di Salmon. In assay troviamo i nomi dei geni (ENSGnumero- ENSG sta per ensembl gene mentre se fosse ENST staremmo lavorando con un trascritto; queste lettere iniziali valgono in umano mentre per altri organismi cambia perché dopo "ENS" viene aggiunto un rimando all'organismo); il numero di gene rimane uguale in tutti gli aggiornamenti a parte il valore dopo il punto che può cambiare perché corrisponde all'esone).
Il Coldata troviamo names, donor e condition: names sono ovviamente i nomi dei campioni, donor la linea cellulare e condition è se il campione è trattato oppure no.
[Riassumendo nel primo passaggio andiamo a caricare la libreria airway e a verificare il contenuto del pacchetto, prendiamo nel file csv le informazioni sui campioni e le carichiamo in una tabella che si chiama coldata; questo è un dataset conformato che ha le sue informazioni confermate ma nel caso di un esperimento nostro questa tabella la scriviamo noi con un editor di testo e carichiamo i dati come summarized experiment. Ora carichiamo tutto il dataset che contiene tutti gli otto campioni del pacchetto airway.]
Siccome stiamo guardando un'analisi di RNAseq utilizziamo il pacchetto DESeq2 che è un pacchetto che permette di fare analisi di differential expression. Per fare questo carichiamo la libreria deseq2 e utilizziamo la funzione DESeqDataSet per trasformare il nostro ranged summarized experiment in un oggetto di DESeq e lo salviamo separatamente. Dobbiamo dare a questo oggetto un design ossia definire quali sono le caratteristiche dei campioni che riteniamo interessanti per successivamente fare ulteriori analisi quindi nel nostro caso avevamo 8 campioni tratti da due linee cellulari ciascuna presente due volte e per ogni linea cellulare avevamo un campione trattato e uno non trattato. Quindi il nostro design DESeq si comunica con una formula dando una tilde e poi tutte le caratteristiche che ci interessano.

```
library("DESeq2")


dds <- DESeqDataSet(gse,
design = ~ donor + condition)
```

A questo punto contiamo il numero di righe presenti in dds; sono un numero elevato di geni e per le nostre future analisi possiamo ridurre la tabella con le conte togliendo i geni che in tutti i campioni non sono espressi perché non ci danno informazioni. Per fare questo possiamo fare la somma dei valori della riga e richiedere che questa sia maggiore di 1 cioè che ci sia almeno una conta per ogni gene. Salviamo questa conta in una variabile che chiamiamo keep [keep<- rowSums (counts(dds)) >1]. Quindi filtriamo il nostro oggetto deseq2 utilizzando soltanto le righe di keep [dds <- dds[keep,]]. Contando nuovamente il numero di geni, ossia di righe di dds [nrow (dds)], il valore è circa dimezzato. Questo passaggio detto di prefiltering serve a possedere solo i geni che ci servono.
Il passaggio successivo è una trasformazione di dati che si chiama Variance Stabilizing Transformation utilizzando una funzione che si chiama VST. Dobbiamo fare questo passaggio perché in genere per le analisi che andremo a fare (come principal contenent analysis e il clustering) è meglio se i nostri dati hanno una varianza circa costante mentre nelle analisi di RNAseq abbiamo una varianza che sale mano a mano che i livelli di espressione salgono. Questa trasformazione ci permette di avere una varianza nello stesso range in tutti i campioni indipendentemente dalla media dell'espressione. Diamo quindi la funzione vst [vsd <- vst (dds, blind=FALSE)]; ci sono altre tecniche per fare questa trasformazione e un esempio è trasformare i dati delle conte con il logaritmo in base due ma anche quello ha delle controindicazioni quindi a seconda

dell'esperimento decidiamo l'approccio migliore. Nel nostro caso è vst perché è anche molto veloce da fare. [Oltre a darci una varianza costante questa trasformazione, normalizza le conte per la library size ed è importante perché quando si sequenzia si ottiene un numero di reads non costante per tutti i campioni (idealmente la variabilità è poca ma dipende da molti fattori) e quindi per uno stesso gene è auspicabile normalizzare anche il numero di reads perché sequenziando più materiale da una parte e meno dall'altra possiamo introdurre un errore.]

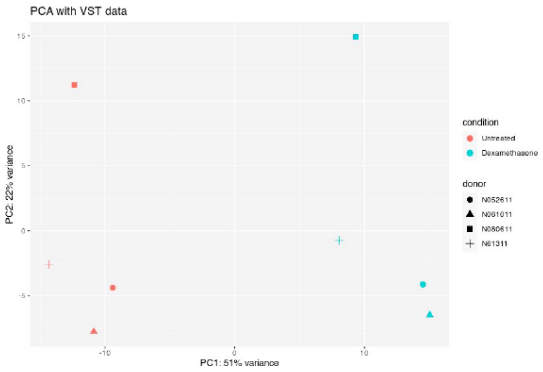
```
pcaData <- plotPCA(vsd,
intgroup = c( "condition",
"donor"), returnData = TRUE)


pcaData



percentVar <- round(100 *
attr(pcaData, "percentVar"))
```

Il passaggio immediatamente successivo è quindi la PCA tramite cui otteniamo le componenti principali e, per ogni campione, ci da la loro localizzazione sul piano bidimensionale. Prima di plottarlo facciamo un ulteriore passaggio che ci permette di calcolare quanta percentuale della variabilità è spiegata dalle principal component e poi questo numero dovremo plottarlo per fare principal component.

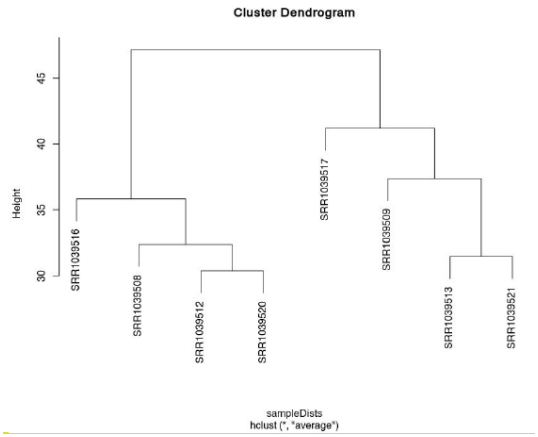
Il secondo passaggio è quindi andare a fare questo plot e per farlo carichiamo la libreria ggplot2 e procediamo a costruire il plot. Di solito la prima principal component va sull'asse x e la seconda sull'asse y, scegliamo il colore il base alla condizione (trattato o no) e la shape dei punti in base alla linea cellulare (donor).

```
library(ggplot2)

ggplot(pcaData, aes(x = PC1, y
= PC2, color = condition, shape
= donor)) +
  geom_point(size =3) +
  xlab(paste0("PC1: ",
percentVar[1], "% variance")) +
  ylab(paste0("PC2: ",
percentVar[2], "% variance")) +
  coord_fixed() +
  ggtitle("PCA with VST data")
```

Nella costruzione del grafico avevamo chiesto di plottare con colore diverso in base al trattamento e abbiamo dato una forma diversa ai punti in base alla linea cellulare. Qui vediamo un plot in cui la principal component 1 sembra pari in trattato e non trattato ed è ragionevole considerando che qui stiamo valutando l'espressione genica nei diversi campioni ci aspettiamo di vedere più distanza e più vicinanza in base a quanto l'espressione varia nei campioni (questa non è un'analisi di diferential expression che ci dice che trattati e non trattati sono espressi in maniera diversa ma un'analisi di riduzione della dimensionalità per cui vediamo come i nostri campioni sono plottati nello spazio in funzione della loro variabilità. Qui facciamo dei ragionamenti molto generali e preliminari per vedere se ci sono problemi nel nostro dataset e vedere se la prima componente principale distingue bene le condizioni; si vede inoltre che una linea cellulare ha un livello di espressione molto diversa rispetto alle altre che stanno tutte molto più vicine (quindi sono meno variabili).



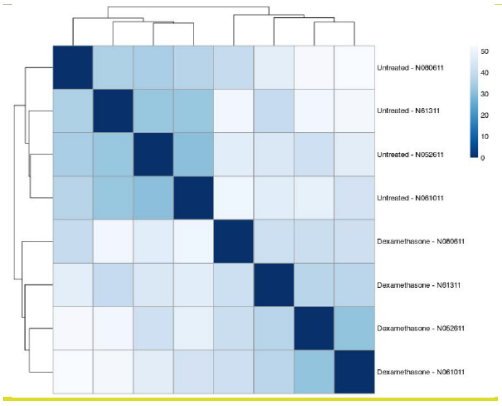
A questo punto calcoliamo le distanze tra i campioni con la misura Euclidea sul dataset vsd. Nella funzione (sampleDists <- dist (t(assay(vsd), method= "euclidean") usiamo il comando t che serve per trasporre la matrice perché in questo caso la funzione dist ha bisogno che i campioni siano sulle righe e i geni sulle colonne. SampleDist, se verifichiamo il contenuto, vediamo esplicitare le distanze tra campioni; utilizzeremo questo per fare il clustering gerarchico con la funzione hclust (hclust_avg <- hclust (sampleDists, method= 'average')).



Possiamo quindi plottare e salvare il plot che, come già aveva fatto il plot, divide nettamente i campioni per condizione e inoltre avvicina le due linee cellulari che avevamo visto essere vicini nella PCA.

Sempre questa matrice di Sample Distances possiamo poi plottarla come heatmap e per farlo dobbiamo caricare le library pheatmap e RcolorBrewer (questa seconda serve semplicemente per modificare i colori

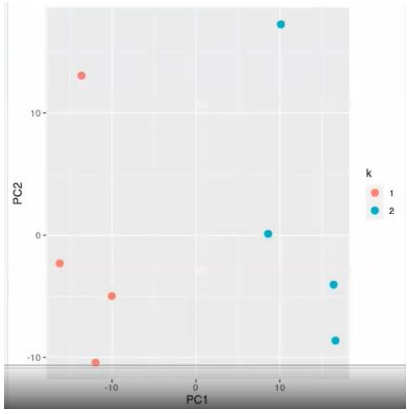
della heatmap). Trasformiamo la tabella sample dists in matrice (sampleDistMatrix <- as.matrix (sampleDists)) e aggiungiamo i nomi delle righe che sono la combinazione della condizione (cioè del trattamento) e della linea cellulare (rownames(sampleDistMatrix) <- paste (vsd$condition, vsd$donor, sep= "-")), non diamo un nome alle colonne, definiamo i colori della heatmap e finalmente possiamo plottare la nostra heatmap con tutti i parametri definiti.



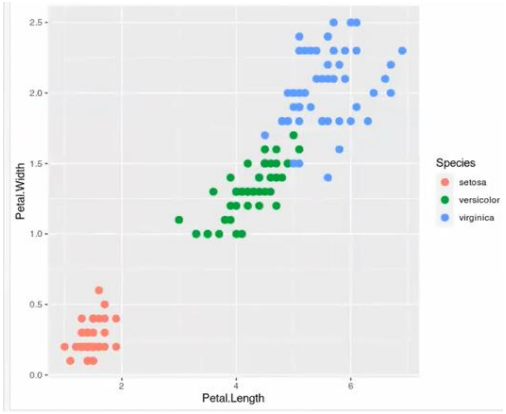
Ci troviamo a vedere la stessa mappa di distanze rappresentata con una mappa di colori (ogni cella è un incrocio tra coppie, il colore più scuro è una vicinanza maggiore) ed è più facile ragionarci sopra; di nuovo qui la grossa separazione è tra trattati e non trattati.
Il prossimo passaggio è quello di provare un clustering, con gli stessi dati, con k-means. Il primo passo per costruire il clustering per k-means è scegliere un valore di k e lo facciamo con la funzione clusters <- kmeans (sampleDists, k). Scegliamo il valore di 2 (quindi i parametri diventano sampleDists e 2) e nell'interfaccia vediamo riportare un clustering vector che assegna i campioni a un gruppo o all'altro. Andiamo a controllare la divisione in gruppi e vediamo anche in questo caso la divisione 4-4 come vedevamo nei precedenti clustering (trattati vs non trattati).
Se invece cambiamo il valore di k e dividiamo i campioni in 4 gruppi otteniamo due gruppi che contengono che contengono campioni trattati e due che contengono campioni non trattati e tra i quattro campioni trattati e non trattati che di dividono in due gruppi di dividono come 3 a 1 perché c'è un campione più separato dagli altri.

```
PCk2 <- pcaData[c('PC1','PC2')]

clustersPC <- kmeans(PCk2, 2)

PCk2['k'] <- as.factor(clustersPC$cluster)

ggplot(PCk2, aes(x = PC1, y = PC2, color =k)) +
+geom_point(size =3)
```

A questo punto facciamo il clustering sulle due componenti principali calcolate con la PCA. Questo ci serve per vedere in un plot i due clusters e per fare questo utilizziamo il dataset bidimensionale che possiamo utilizzare per plottare e dividere i campioni per colore a seconda della loro appartenenza a un gruppo o all'altro.

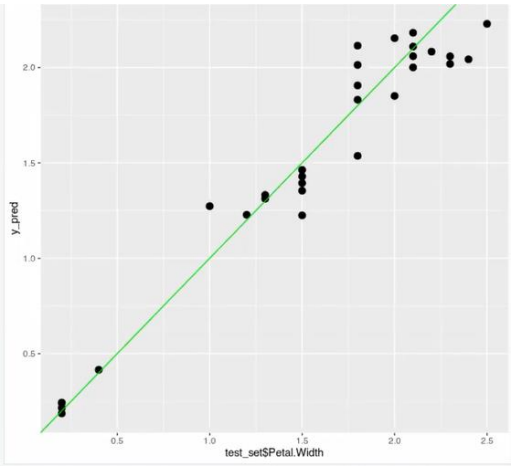


Il plot è di nuovo quello della PCA ma questa volta i colori corrispondono al gruppo quindi indica l'appartenenza ad un dato k. Questo riesce a darci un'idea visuale della suddivisione vicino ai centroidi.
Abbiamo terminato la parte del clustering e andiamo avanti con il supervised learning utilizzando, per questo, un dataset che si chiama iris e che contiene molti più campioni del precedente. Questo dataset contiene informazioni sui fiori e le differenze fra diverse specie e ogni campione presenta lunghezza e larghezza dei petali e dei sepali. Precisamente la tabella presenta le prime quattro colonne con le caratteristiche fisiche appena elencate e la quinta colonna è il nome della specie. Tutto il dataset è diviso in tre specie con cinquanta campioni per specie. Possiamo fare un primo plot inserendo in x la lunghezza del petalo e in y la larghezza degli stessi e colorare ogni specie in maniera diversa; vediamo chiaramente che i tre gruppi hanno misure simili di larghezza e lunghezza dei petali.



Cominciamo applicando analisi di regressione lineare sul nostro dataset; per cominciare dividiamo il nostro dataset in due gruppi che sono il training set e il set di test. Per fare questo utilizziamo una funzione che si chiama sample (ran <- sample (1:nrow(iris), 0.8*nrow(iris)) con cui diciamo di prendere su un vettore che contiene tutte le

righe di iris l'80% dei dati che andrà a formare il nostro training set. Invece nel test set inseriamo tutte le righe restanti (test_set=iris[-ran,]). Come abbiamo già detto il set di training ci serve per stimare la regressione lineare e il set di test ci serve per verificarne la statistica. Per fittare un modello di regressione lineare sul set di training usiamo la funzione lm (rg = lm (formula =Petal.Width~., data=training_set) cioè stiamo chiedendo di fare la regressione della larghezza dei petali rispetto a tutti gli altri valori). Data la regressione ora andiamo a predire i valori (y_per = predict (reg, newdata=test_set) che poi verranno confrontati con il test set per verificare quanto la predizione si avvicina ai dati reali; un primo metodo per farlo è la differenza tra i valori predetti e i valori osservati e vedere quanto questo sia vicino a zero (quindi facciamo la distribuzione dei risultati delle differenze con cui viene fuori un istogramma con tanti valori vicino a zero e qualche valore più distante; per avere un output grafico possiamo fare uno scatterplot plottando sulle y le predizioni appena fatte e sulla x i dati originali di larghezza dei petali. Se la differenza è zero o molto vicina a zero i nostri punti si disporranno lungo la diagonale (test_pred <- data.frame (test_set$Petal.Width, y_pred) e poi plottiamo i dati con il ggplot in cui oltre a plottare i nostri dati plottiamo anche una diagonale con intercetta 0 e pendenza uguale a 1).



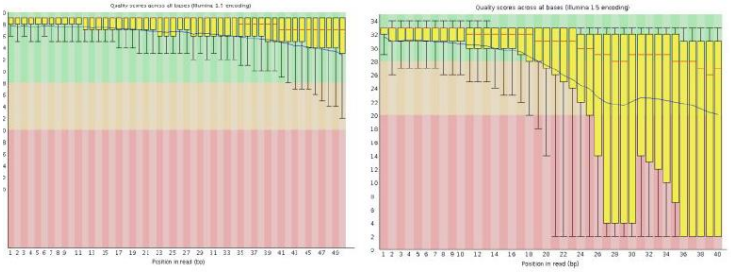
Viene fuori un plot abbastanza buono con alcuni punti sulla retta e alcuni punti poco lontani dalla stessa. Questo era abbastanza scontato anche perché per il modello usiamo tutti i dati a disposizione (se provassimo a predire la petal width usando solo la petal length la predizione andrebbe peggio).

Usiamo poi la k-nearest neighbors continuando a usare il dataset iris; utilizzando questo dataset possiamo, viste le misure fenotipiche, dividere i campioni in base alla specie cioè assegnare la specie date le misure fenotipiche. Dobbiamo normalizzare i dati perché è meglio avere i dati nello stesso range in modo che se ci sono campioni con valori estremi questi pesano meno (osservando il summary ci rendiamo conto che con questo dataset non sarebbe proprio necessario fare la normalizzazione perché minimi e massimi di tutti i caratteri sono più o meno nello stesso range). Per farlo creiamo una funzione che chiamiamo norm (norm <- function (x)) e dobbiamo fare il rapporto tra numeratore e denominatore ma prima definiamo numeratore e denominatore dove il numeratore equivale alla distanza di ciascun valore dal minimo e il denominatore è la distanza del massimo dal minimo (num <- x-min(x) e denom <- max(x) -min(x)). Definiamo il nuovo dataset iris_norm tramite la funzione apply che ci permette di applicare la funzione norm da noi creata a tutti gli elementi di iris ma ovviamente non vogliamo applicare a tutte le colonne ma alle prime 4 (visto che l'ultima sono i nomi) quindi in stringa iris_norm <- as.data.frame (lapply(iris[1:4], norm)). Vedendo il nuovo summary i valori di ogni colonna diventano compresi tra 0 e 1.

Per applicare KNN dobbiamo dividere il dataset in due gruppi (ancora training e test) e lo facciamo in modo leggermente diverso rispetto a prima: utilizziamo sempre la funzione Sample che ad ogni riga di iris affida un numero tra 1 e due e lo fa utilizzando il replacement (se la prima riga affida il numero 1, il vettore da cui pesco i valori che contiene 1 e 2 non viene svuotato ma continua a contenere i valori in modo da poter affidare lo stesso valore). Per far chiarezza dobbiamo dividere il dataset in due gruppi e invece semplicemente di dividere chiedendo la prima metà o il primo 30%, siccome i campioni sono ordinati per specie e non possiamo fare un training set che contenga la prima specie e il test set che contenga la terza me dobbiamo avere dei mix, usiamo una funzione che per ogni riga del dataset originale assegni un valore casuale che viene da un vettore che contiene i numeri 1 e 2; usiamo il replacement per far sì che il valore assegnato non infici la probabilità di assegnare il valore e vogliamo che questo sia indipendente dalle assegnazioni precedenti. In più vogliamo che il valore 1 abbia una probabilità del 67% e il valore 2 del 33% in modo che il nostro training set sia più grande del nostro test set. Siccome l'affidamento del numero è random, usiamo un numero seed [set.seed (1234) ; ind <-sample(2, nrow(iris), replace=TRUE, prob=c (0.67, 0.33)]. Creiamo sulla base di questo vettore i nostri set di training (iris.training <- iris[ind==1, 1:4]) e di test

(iris.test <- iris [ind==2, 1:4]) e, per assicurarci che siano ben distribuiti, andiamo a controllare le dimensioni con una funzione che si chiama dim (). Ora però dobbiamo raccogliere le informazioni che riguardano la specie perché nel dataset di training ora abbiamo soltanto i valori ma se il training ci serve per imparare in base a quali valori determiniamo la classificazione della specie dobbiamo avere questa informazione (iris.trainLabels <- iris [ind==1,5] e iris.testLabels <- iris [ind==2, 5]). A questo punto carichiamo il pacchetto che ci permette di utilizzare la classificazione con KNN (library (class)) e salviamo la nostra predizione in iris_pred (iris_pred <- knn (train=iris.training, test=iris.test, cl=iris.trainLabels, k=3). Nelle nostre predizioni ora sono contenuti i nomi delle specie. Possiamo quindi valutare le predizioni andando a confrontare le specie predette con quelle del dataset originale: per farlo trasformiamo i nostri datatest in un dataframe (irisTestLabels <- data.frame (iris.test.Labels), creiamo il dataframe merge in cui inseriamo le predizioni e i nostri valori osservati (merge <- data.frame (iris_pred, irisTestLabels), nominiamo le colonne del dataframe merge così sappiamo cosa stiamo guardando (names (merge) <- c("Predicted Species", "Observed Species") e generiamo il merge con cui osserviamo se il training è fatto correttamente e la predizione funziona, quindi se le specie predette corrispondono alle osservate. Effettivamente il modello è abbastanza buono, solo due campioni sono mislabeled e questo ci dice che effettivamente il fenotipo è distribuito in maniera sufficientemente diversa tra le specie da permetterci di distinguerle. (Facendo un plot per petali e per sepali ci rendiamo conto che la divisione in base ai petali è più significativa e caratterizzano maggiormente le classi; i valori confusi si possono spiegare in questo modo, con la caratterizzazione dei sepali che un po' si sovrappone).

Pipeline in high-throughput biology. Ripartiamo dagli esempi di esperimenti di analisi del trascrittoma. Abbiamo già visto i primi pezzi e ora ripercorriamo l'intero processo un pezzo alla volta. Una volta che abbiamo disegnato il nostro design sperimentale e fatto l'esperimento nel nostro sistema modello, quindi estratto l'RNA, costruito le libraries e sequenziato, otteniamo le nostre reads di sequenziamento grezze in formato FASTQ. In generale le reads vengono filtrate in base alla loro qualità: quando abbiamo provato a fare l'allineamento noi le abbiamo prese così com'erano perché lavoravamo su un dataset di esempio già filtrato ma quando abbiamo il nostro esperimento nuovo dobbiamo fare il controllo di qualità iniziale sulle reads. La qualità delle basi è codificata da uno score rappresentato nella quarta riga del FASTQ di ogni read e questo score rappresenta la probabilità che la chiamata di quella base sia un errore. Oltre alla qualità nelle reads controlliamo la presenza di contaminanti prima di allineare e questo si fa con un tool che si chiama FastQC (Fast Quality Control) e ci da un plot dell'output della qualità.



Good quality!

Poor quality

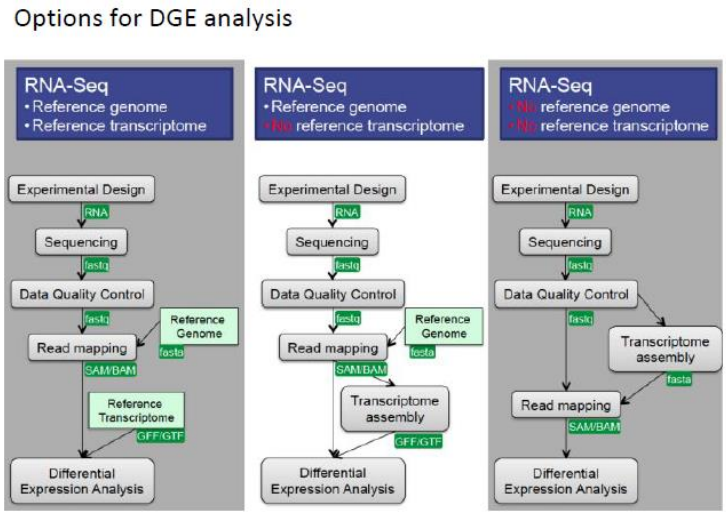
Lungo l'asse x troviamo la posizione delle basi nella read e in questo caso abbiamo delle read lunghe 50 paia di basi mentre sull'asse y è presente il phred score che è diviso in fasce per cui sotto il venti la qualità è molto bassa, segue una fascia intermedia e dal 28 in su la qualità è molto buona; un phred score di 30 ci dice che abbiamo una probabilità su mille che la base chiamata sia un errore mentre un phred score di venti è una probabilità su 100 e di 10 è una probabilità su dieci. In questo plot sono presenti dei box plot che rappresentano tutte le reads sul nostro campione; la linea presente a metà di ogni rettangolino dei boxplot rappresenta la mediana. Nel campione di destra vediamo come la qualità scenda pesantemente all'aumentare del numero di reads e questo è problematico perché in questo campione la qualità da metà reads in avanti è talmente bassa che non ci possiamo fidare di dove la reads allinei. FastQC oltre a darci il plot della qualità di tutte le reads legge tutti i fastQ leggendo la qualità di ogni read e ci fa un plot di summary di tutte le reads presenti in quel campione. (In dettaglio ogni rettangolo del plot rappresenta una posizione e quindi quanta qualità occorre in tutte le reads del campione in ogni posizione quindi ad esempio in posizione 35 controlla la possibilità di errore, che la chiamata della coppia di basi sia sbagliata,

basandosi su tutte le reads del campione.) Oltre alla qualità, che in un certo senso è solo una visualizzazione grafica della qualità riportata in quarta posizione del fastQ cioè all'espressione della qualità del nucleotide, fastQC controlla anche altri aspetti come la presenza di sequenze contaminanti, la lunghezza media delle reads e il contenuto di GC. Ci sono anche altri parametri sempre più specifici che vanno valutati sempre a seconda di come è fatto l'esperimento, dell'organismo su cui stiamo lavorando e di come è stata preparata la library. Quello che fastQC fa è darci una valutazione espressa come i colori di un semaforo: il verde rappresenta che va tutto bene, arancione da un warning, quindi attira l'attenzione su punti che potrebbero essere critici e poi segna in rosso eventuali fallimenti; FastQC riesce a fare questo perché al suo interno ha delle soglie, sia per la qualità che ad esempio per la presenza di sequenza contaminanti, che sono ben definite e noi non possiamo prendere come sentenza finale quello che dice fastQC ma, come abbiamo visto con altri tool, dobbiamo prendere l'informazione che il software ci dà e valutarla in base a tutte le caratteristiche del nostro esperimento e quindi decidere se le reads vanno bene nonostante la qualità non sia ottima oppure rigettarle. Una volta controllata la qualità delle nostre reads, nel caso ci sia ad esempio una bassa qualità o nel caso della presenza di contaminanti. Generalmente la contaminazione può essere data da sequenze estranee all'organismo che stiamo studiando (come se stiamo studiando l'uomo e troviamo sequenza non umane) ma non viene identificata in questo punto; ci si accorge che abbiamo reads contaminanti quando andiamo ad allineare perché avremo un reference in cui molte nostre reads non allineeranno. La contaminazione riconosciuta in questa fase è una contaminazione da adattatori che sono delle sequenze utilizzate per fare le libraries; quindi può capitare a volte che questi adattatori vengano sequenziati e quindi risultino nelle nostre reads. Per risolvere questa cosa semplicemente si eliminano le sequenze degli adattatori dalle reads; ci sono dei tool che permettono di tagliare le reads partendo dalle estremità, finché la qualità è bassa quindi noi ci diamo delle soglie che sono ad esempio un phred score di 28 o 26 e fino a che il tool non incontra una base con qualità 26 taglia e rimuove tutta la parte della read con una bassa qualità. È chiaro che avendo una read più corta perdiamo dell'informazione ma se questa informazione non è affidabile, come nel caso di una qualità bassa, non ce ne faremmo niente e quindi può essere solo un vantaggio rimuovere parti di sequenza tramite software che fanno trimming delle reads (lo stesso può essere fatto per la rimozione degli adattatori; in più però gli adattatori presentano sequenze note e riconosciute per cui i software di trimming possono eliminarle completamente dalle nostre reads). Successivi a questi passaggi di pulizia e di trimming delle sequenze, rifacciamo girare fastQC per vedere come e se migliora a qualità delle nostre basi. Se tagliamo le nostre sequenze di molto risulteranno troppo corte per avere degli allineamenti unici e quando una read allinea troppe parti non riusciamo a mappar la sua provenienza sul genoma e quindi posso decidere se scartarla o considerare in maniera diversa l'informazione proveniente da quella read. Inoltre quando abbiamo visto i file fastQ che contenevano reads paired end quindi un fastQ per le read forward e una per le reads reverse, in generale i tool di allineamento vogliono che le reads nel file 1 e nel file 2 siano accoppiate; se nella mia coppia di reads hoeliminato o la forward o la reverse perché la read aveva qualità molto bassa, questo poi può creare problemi all'allineatore perché non trova la seconda parte della coppia della reads e in questo caso, ancora, si filtrano le reads per le coppie che sono rimaste tali e invece le reads rimaste singole perché la loro coppia è stata eliminata vengono allineate successivamente come single end e l'allineamento viene considerato poi unico.

Quindi la prima fase comprende controllo qualità, trimming e poi un secondo controllo qualità delle reads grezze; può seguire quindi l'allineamento o quantificazione che abbiamo visto sia in forma di allineamento che di quantificazione diretta dei trascritti (lo abbiamo già visto nelle esercitazioni precedenti, è solo per mantenere il flow degli esperimenti).

Il passaggio successivo nel nostro esperimento di analisi del trascrittoma è l'analisi di espressione differenziale posto che abbiamo due condizioni (che nel nostro esempio erano linee cellulari trattate e non trattate) e quindi procediamo all'analisi di espressione differenziale. Ci sono più possibilità di workflow come opzioni per esperimenti di tipo diverso sempre di espressione differenziale ma in questo caso abbiamo un genoma e un trascrittoma di riferimento. Il genoma è stato sequenziato, assemblato e abbiamo

un genoma di riferimento e abbiamo anche un'annotazione dei trascritti quindi le sequenze dei trascritti e la loro struttura sono disponibili e in questo caso sono possibili entrambi gli approcci visti che sono il mapping e lo pseudomapping; partendo dall'RNA abbiamo il passaggio del sequencing con cui otteniamo il fastQ, il data quality control lo abbiamo appena visto, abbiamo visto anche il mapping che ci permette di ottenere un file di allineamento SAM o BAM e da qui, poi, passando per il trascrittoma di riferimento otteniamo le conte per ciascun gene oppure saltando il passaggio di allineamento sul genoma (quindi tramite lo pseudomapping di Salmon) otteniamo comunque le quantificazioni che sono il nostro input per misurare l'espressione differenziale (e questo è il caso più semplice e diretto).
È possibile poi che per l'organismo su cui stiamo lavorando sia disponibile un genoma di riferimento ma non un trascrittoma (cioè è possibile che il genoma sia stato sequenziato e assemblato, la sequenza sia disponibile ma non sia disponibile nessuna predizione sui trascritti per varie ragioni); in questo caso i passaggi a monte, fino al mapping, sono gli stessi ma a valle abbiamo un passaggio di assemblaggio di trascritti prima di arrivare alle conte e questo è fatto con dei tool appositi per l'assemblaggio dei trascritti partendo dalle reads del nostro RNA-seq.



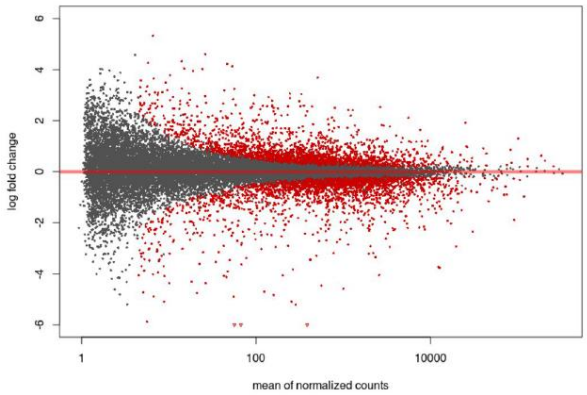
Il terzo caso è quello in cui non abbiamo come reference né un genoma né un trascrittoma e quindi direttamente prima di mappare sul genoma si fa un assemblaggio dei trascritti, per quanto sia possibile, e poi sull'assemblaggio dei trascritti si fa la quantificazione (prima le reads vengono usate per assemblare il trascrittoma poi vengono allineate al trascrittoma per ottenere le conte).
Fatta questa anticipazione partiamo con il classico esperimento di espressione differenziale: carichiamo le due libraries che ci servono (DESeq2 e airway) e i dati (airway) e creiamo il DESeq dataset (dds <- DESeqDataSet (airway, design = ~ cell+dex)); lo filtriamo, come avevamo visto, tenendo i geni che con nessuno di tutti i campioni hanno almeno 10 reads e questo ci serve sia perché i geni che non hanno espressione non sono informativi (quindi possiamo eliminarli visto che comunque seguono soglie molto stringenti per avere un minimo livello di espressione in un gene in un campione) sia perché così rendiamo un po' più leggero il nostro dataset e siamo più efficienti a livello di memoria (keep <-rowSums(counts(dds)) >= 10; dds <- dds[keep,]). La precedente normalizzazione che avevamo fatto ci permetteva di ottenere un dataset con una varianza costante rispetto al livello di espressione; ora questo non lo abbiamo ancora fatto e abbiamo ancora i dati grezzi in dds perché DESeq2 per fare analisi di espressione differenziale vuole le conte grezze (perché all'interno della funzione di analisi differenziale la normalizzazione viene fatta internamente e cioè vengono utilizzati i dati grezzi perché servono per eseguire la normalizzazione e calcolare i parametri per il calcolo del differenziale). Quindi è da tenere a mente che l'input dell'espressione differenziale sono le conte grezza ma quando noi facciamo un'analisi preliminare tipo la PCA trasformiamo i dati e li teniamo trasformati.
Fatte queste considerazioni possiamo avviare la vera e propria analisi di espressione differenziale che si lancia con la funzione DESeq e la salviamo all'interno di dds (dds <- DESeq (dds)); quello che questa funzione fa è stimare i size factors che sono la sequencing depth che è la profondità di sequenziamento per ogni campione e la media di espressione per ogni gene attraverso i campioni; inoltre viene calcolata la extimating dispersion che in questo caso è la variabilità tra i replicati . Quando io confronto due condizioni e ho dei replicati per ciascuna condizione, in una situazione ideale i replicati sono concordi nel senso che hanno un livello di espressione molto simile in entrambe le condizioni quindi il cambiamento che vediamo tra le condizioni idealmente è supportato dai tre replicati in ciascuna condizione; se il mio gene di interesse

è espresso a livello molto basso nei controlli e nei trattati è altamente espresso solo in un replicato, faccio fatica a identificarlo come differenzialmente espresso perché è più logico pensare che magari sia successo qualcosa in quel campione o che quel campione abbia una variabilità molto grande e che quindi quel gene sia espresso in maniera maggiore in quel replicato per vari motivi tecnici. È importante valutare la variabilità tra i replicati per avere poi un risultato significativo. Questi parametri poi vengono fittati in un modello a cui poi viene fatto il test per la significatività e questa poi viene ulteriormente corretta per il multiple testing. Utilizziamo quindi la funzione results per estrarre i risultati del dds (res <- results (dds)).

	baseMean	log2FoldChange	lfcSE
	<numeric>	<numeric>	<numeric>
ENSG00000000003	708.597861536998	0.381227158187308	0.100702283808889
ENSG00000000419	520.296296925274	-0.206840262096018	0.112107726739565
ENSG00000000457	237.162103834464	-0.0379541749180647	0.14282308754538
ENSG00000000460	57.9323803212894	0.0885313640426604	0.284934405702223
ENSG00000000971	5817.31081674539	-0.426424458656131	0.0888056168163761
...	...	...	...
ENSG00000273483	2.68955174874763	-0.849207778045454	1.25336481271555
ENSG00000273485	1.28646279725438	0.12361364195379	1.58825007006569
ENSG00000273486	15.4524429107135	0.150429500144829	0.482097669094512
ENSG00000273487	8.16326862804303	-1.04563791009899	0.693057055280173
ENSG00000273488	8.58437098976254	-0.108944574451661	0.632299672638331

	stat	pvalue
	<numeric>	<numeric>
ENSG00000000003	3.78568532676772	0.0001532855453986
ENSG00000000419	-1.84501343584037	0.0650355841605804
ENSG00000000457	-0.265742574049909	0.790437472527865
ENSG00000000460	0.310707876167058	0.756022709110766
ENSG00000000971	-4.80177351324356	1.57266511553479e-06
...	...	...
ENSG00000273483	-0.677542379864291	0.498061890561127
ENSG00000273485	0.0778300875180678	0.937963212655999
ENSG00000273486	0.31203117083592	0.75501683424151
ENSG00000273487	-1.50873279787373	0.131367080419078
ENSG00000273488	-0.17229895754505	0.863202503856328

In questo file vedremo per ogni riga un gene e le colonne rappresentano le baseMean (media delle conte normalizzate su tutti i campioni), il log2FoldChange (che esprime di quanto cambia l'espressione genica in una condizione rispetto che in un'altra), lfcSE (lo standard error associato al log2FoldChange), lo stat (output della statistica differenziale), p-value e padj (adjusted p-value). Stiamo confrontando la nostra condizione dex con la condizione untreated e i p-value si riferiscono a questo confronto quindi quando il nostro log2FoldChange è negativo vuol dire che abbiamo una downregolazione nell'untreated. In basso, nella colonna padj, per dei valori segnala "NA"; se nella colonna baseMean c'è uno 0 vuol dire che tutti i campioni per quel gene hanno espressione 0 e quindi avremo un NA in tutte le altre colonne perché le statistiche non sono applicabili ma questo non è il nostro caso perché quelle righe le abbiamo già eliminate tutte con il filtering. È possibile che p-value e padj abbiano degli NA in geni che sono outliers estremi per quanto riguarda l'espressione e cioè hanno un'espressione molto molto alta o bassa che è talmente estrema che non fitta il modello e quindi viene eliminata perché non darebbe un risultato affidabile; invece quando abbiamo un NA nella colonna padj vuol dire che la media delle conte è bassa e non possiamo avere una statistica affidabile.



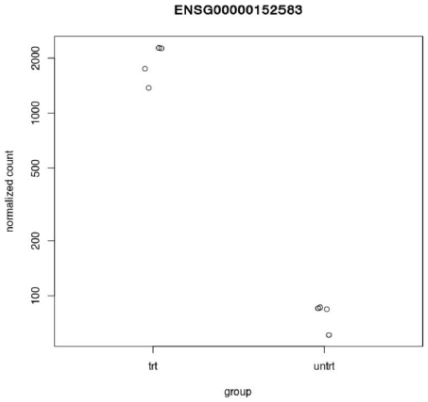
Ora andiamo ad esplorare questi risultati facendo un plot MA che plotti log2FC sull'asse y e la media delle conte normalizzate sull'asse x (plotMA (res, ylim=c(-6,6))). Ogni puntino è un gene, i geni sono colorati in grigio e in rosso: in rosso sono rappresentati i geni considerati significativi in base al p-value (tra le varie cose tra i parametri del plot MA possiamo cambiare anche la soglia oltre il quale questi plot vengono segnalati come significativi). Notiamo infatti che i geni espressi a basso livello anche con un FoldChange alto non sono considerati significativi perché è il p-value a darci una dimostrazione della significatività. (Quando il livello di espressione basale è basso ci vuole molto poco per ottenere un aumento del fold change perché questo identifica quante volte il gene è espresso rispetto alla condizione normale mentre se il livello di espressione basale è alto il logfoldchange fa più fatica ad aumentare quindi è importante calcolarne la significatività tramite il p-value).

Questo ci dà già una buona idea di come sono distribuiti i nostri geni rispetto al fold change; vediamo che ci sono dei cambiamenti tra le due condizioni (quindi questo farmaco ha un effetto sulle nostre cellule).

Un altro modo per valutare ed esplorare i nostri risultati è quello di plottare le conte, quindi l'espressione, per condizione di un gene alla volta. Possiamo ad esempio andare a prendere il gene che nei nostri risultati ha padj più basso [plotCounts (dds, gene=which.min(res$padj), intgroup="condition")].

Nel nostro caso il gene padj minore è SPARCL1 (lo cerchiamo su ensemble a partire dal nome che viene plottato sul grafico; ovviamente ricordare gli emsemble ID è difficile già per i geni più comuni ed è impossibile ricordarne la corrispondenza di un alto numero quindi un passaggio successivo che faremo nelle

analisi sarà quello di aggiungere il gene symbol e la descrizione- ad esempio su che tipo di gene è, come un microRNA o un protein-coding o un long non coding; queste informazioni, dato l'ensemble ID le nostre annotazioni le possiamo trovare nel file GTF o GFF, file scaricabile dal genome browser, da ensemble oppure quando scarichiamo i nostri reference abbiamo i fasta disponibili per human e mouse in gencode e lì ci sono anche le annotazioni).



Avendo selezionato noi il file in quanto file con padj minore sappiamo già che è significativo; dal grafico vediamo poi in maniera molto evidente che in tutti i campioni è espresso molto poco nell'untreated e molto nel treated e quindi deduciamo, di nuovo, che il trattamento causa una maggiore espressione di questo gene; con questa informazione cerchiamo poi di capire che gene sia e le possibili conseguenze dell'upregolazione di questo gene.

Ci serve guardare il plot delle conte per verificare che l'espressione sia diversa tra le due condizioni ma è chiaro che non possiamo fare questo lavoro con tutti i geni che ci troviamo ad analizzare ma soprattutto non ha senso andare a guardarli uno alla volta (in primis perché ci metteremmo troppo tempo e in secundis operare queste operazioni ripetitive provocherebbe l'introduzione di errori).

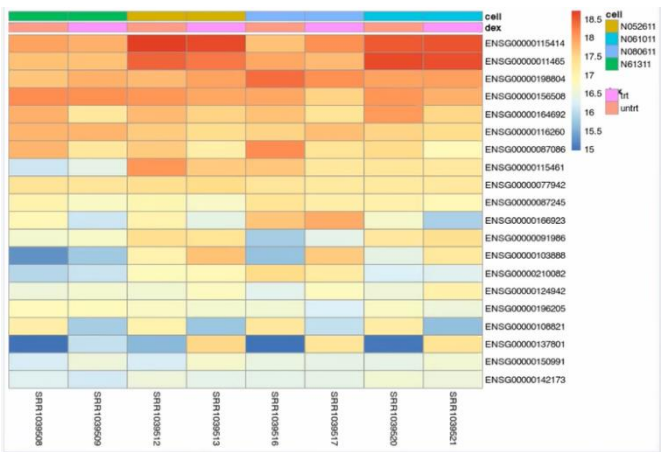
[Qui stiamo facendo l'esempio dell'espressione differenziale ma questo tipo di ragionamenti si può fare anche per altri tipi di esperimenti come la CHiP seq in cui abbiamo delle reads che mappano su alcune coordinate che possono essere i geni o possono essere delle regioni come i promotori e gli enhancer o delle regioni a nostra scelta quindi invece di andare a misurare l'espressione con il numero di reads che mappano lì possiamo misurare la presenza quantitativa di un fattore di trascrizione. Questo può essere fatto con gli stessi metodi cioè contiamo le reads che mappano in una regione nelle due condizioni e posso applicare lo stesso tool o dei tool molto simili per valutare la significatività della differenza fra le due condizioni e in questi tool è sempre disponibile l'opzione per la correzione del multiple testing.]

```
library("pheatmap")

vsd <- vst(dds, blind=FALSE)

select <- order(rowMeans(counts(dds,normalized=TRUE)),
                decreasing=TRUE)[1:20]

df <- as.data.frame(colData(dds)[,c("dex","cell")])

pheatmap(assay(vsd)[select,], cluster_rows=FALSE,
         show_rownames=FALSE, cluster_cols=FALSE,
         annotation_col=df)
```

Per un'ulteriore esplorazione dei nostri risultati possiamo andare a vedere come si comportano tramite una heatmap. Carichiamo la stessa library utilizzata la volta precedente (pheatmap) e per visualizzare in una heatmap le conte utilizziamo, come l'altra volta, quelle normalizzate (vsd <- vst (dds, blind=FALSE)). Con il passaggio successivo andiamo a prendere soltanto alcuni geni e li ordiniamo in ordine decrescente (ci serve soltanto per avere un'idea della possibile visualizzazione).

Prendiamo le colonne che ci interessano da dds e le salviamo in un dataframe; avendo filtrato i dati possiamo quindi già costruire la nostra heatmap.

Con questi soli dati otteniamo una heatmap che abbia lungo le colonne i nostri campioni, una annotazione in alto delle nostre linee cellulari e dei campioni distinti per colori (ci permette di distinguere a occhio le condizioni); in questo subset vediamo pochi geni espressi in maniera decrescente. Serve solamente per vedere graficamente l'espressione di un gruppo di geni, un output grafico ma non ci dice niente dei geni che cambiano tra le due condizioni di cui, comunque, stiamo vedendo un subset molto piccolo. Spesso nelle pubblicazioni quando c'è un esperimento di espressione differenziale vediamo una heatmap, con clustering o meno, e in generale a seconda del numero di geni differenziali, gli stessi vengono descritti o no sugli assi (fattibile quando i geni sono pochi).

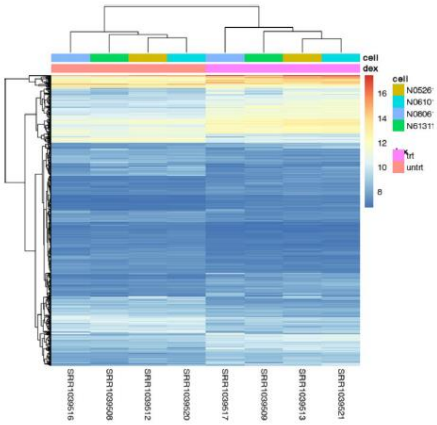


Abbiamo preso in questo caso un subset casuale ma può succedere nei casi specifici che si sia interessati a un sottogruppo di geni esaminati, come i geni di un certo pathway, e si può scegliere di andare a vedere i livelli di espressione. Questo però mostra i livelli di espressione ma non ci dice se sono differenzialmente espressi e quanto cambia fra le due condizioni. Questo mostra semplicemente il livello di espressione di un set di geni.

Questi dati ricavati per questa heatmap quindi sono utili solo per dare questo output grafico. Per mostrare l'espressione differenziale recuperiamo l'output che abbiamo conservato alcuni passaggi fa in res e dove sono contenuti il Log2FoldChange, il p-value e il padj; in aggiunta abbiamo le conte grezze in dds e le conte normalizzate in vsd.

```
sig = res[which(
          abs(res[,"log2FoldChange"])>1
          & res[,"padj"]<=0.05),]


pheatmap(assay(vsd)[rownames(sig),], cluster_rows=TRUE,
          show_rownames=FALSE, cluster_cols=TRUE,
          annotation_col=df)
```

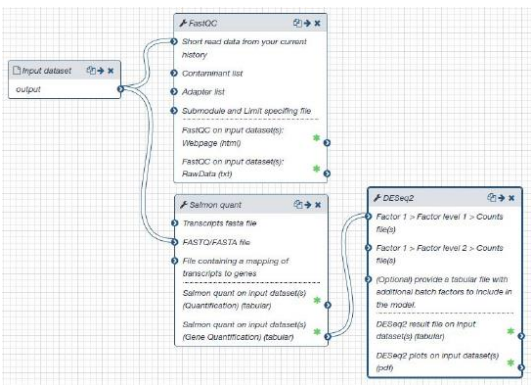
Segue quindi una selezione dei geni significativi. Salviamo nell'oggetto sig alcuni geni di res scelti arbitrariamente cioè quelli che hanno un padj minore o uguale a 0.05 e un valore assoluto di log2FoldChange maggiore di 1 (quindi filtriamo non solo sull'adjusted p-value ma richiedo che ci sia un cambiamento minimo di espressione tra le due condizioni che corrisponde a un raddoppio o a un dimezzamento (perché è a ±1). Guardando la dimensione di res (sempre con il comando dim) scopriamo che i geni sono quasi 23000 mentre i geni significativi, che vediamo nella heatmap, sono solo circa un migliaio. Andiamo a plottare quindi la nostra heatmap dei geni che hanno una variazione significativa (non plottiamo i nomi dei geni perché sarebbero troppi e non leggibili, chiediamo invece di plottare il clustering). Quello che otteniamo è una tabella che ci mostra un confronto diretto tra i campioni trattati e i campioni non trattati (tutti i trattati a destra e i non trattati a sinistra e clusterizzando anche le righe i geni vengono ordinati in base alla loro espressione e non come prima in ordine di espressione decrescente). Questo è il classico output di un esperimento di espressione differenziale e a livello biologico noi sappiamo che in queste linee cellulari trattate in questa maniera otteniamo, con filtro 1 e filtro 2, un migliaio di geni differenzialmente espressi. Chiaramente a livello biologico questo ci dice che questo farmaco ha un impatto sull'espressione genica di queste cellule ma non dice molto di più e quindi il passaggio successivo sarà effettuare l'annotazione funzionale di un set di geni che nel nostro esempio sono i geni differenzialmente espressi.



Abbiamo visto una parte di una pipeline di analisi di un esperimento di RNAseq per espressione differenziale e il passaggio ultimo o che comunque ci da una informazione biologica su cui possiamo fare dei ragionamenti è quello di annotazione funzionale quindi andiamo a prendere dai file di annotazione funzionale il gene symbol, il biotype dei nostri geni (se sono coding o non coding ecc) e andiamo a vedere la loro funzione biologica e faremo quindi delle analisi di annotazione funzionale o di arricchimento per capire quali funzioni biologiche e processi biologici questi geni impattano e lo vedremo le prossime volte.

Oltre a questo set di passaggi possiamo costruire un workflow per un esperimento di questo tipo su Galaxy che mi permette di generare automaticamente pipeline.

Una volta creato l'account possiamo costruire un workflow grazie ai tool che sono elencati sulla sinistra del software (in una finestrella a scorrimento abbiamo la maggior parte dei software e dei tool che usiamo in locale per svolgere i diversi esperimenti). Per creare il workflow semplicemente clicchiamo su di un tool e lo trasciniamo nella finestra di lavoro e poi creiamo dei collegamenti tra i diversi tool. Il vantaggio di usare il workflow rispetto a eseguire manualmente ogni esperimento è sicuramente la riproducibilità; il pregio di Galaxy è la condivisibilità dei workflow.



Biological Database

Visualizziamo alcuni database. Cominciamo con ensembl Vediamo su ensembl un menu a tendina con elencate tutte le specie presenti sul browser e al centro vediamo gli organismi modello ma possiamo accedere all'intera lista delle specie disponibili; ensembl è dedicato in maniera particolare ai vertebrati però è disponibile il sito ensembl genomes, sito collegato che contiene i genomi di batteri, funghi, piante eccetera quindi a seconda della specie di interesse ci dirigiamo verso l'uno o verso l'altro (ma funzionano in maniera simile). Clicchiamo sull'organismo di interesse e ci indirizza ad una pagina con tutte le risorse disponibili: abbiamo la versione del reference di riferimento con disponibile la sequenza in Fasta da scaricare, una pagina di analisi comparative tra più specie, una parte di regolazione genica, l'annotazione (Fasta delle proteine o dei geni o delle coding sequencies scaricabile) e il GTF dell'annotazione genica; poi c'è una sezione dedicata alle varianti geniche.
Utilizziamo un gene di esempio per andare a vedere cosa contiene ensembl (possiamo cercare tramite gene symbol o tramite ensembl ID) e vediamo che tra i risultati della ricerca abbiamo dei trascritti e il gene; se clicchiamo sul gene ci indirizza alla pagina dedicata. Abbiamo una descrizione generale del gene e dei suoi sinonimi, la localizzazione sul cromosoma; il valore in alto che affianca il gene symbol è l'identificativo stabile di ensembl e questo vuol dire che questo gene, indipendentemente dalla versione dell'annotazione, rimane immutato. Nella pagina principale, sulla destra in alto dice quale a release di ensembl siamo e anche dettagli sulle cose aggiornate rispetto all'ultima volta. Torniamo al nostro gene di esempio, ci si apre una lista dei trascritti. Sulla sinistra invece c'è un indice con varie opzioni: c'è il summary che è la prima pagina che si apre e come sottoelenco possiamo vedere diverse cose come varianti di splicing e comparazioni tra trascritti; c'è la sequenza del gene, ci si apre la visualizzazione in cui sono evidenziati gli esoni e questa sequenza è scaricabille o addirittura la si può blastare direttamente. Successivamente abbiamo una parte di genomica comparativa in cui possiamo vedere allineamenti con altre specie: possiamo vedere il gene tree cioè un albero con tutte le specie presenti all'interno di ensembl e l'allineamento delle sequenze proteiche di cui vediamo disegnati i domini. L'albero può essere esplorato in ogni sua parte andando ad espandere o rimpicciolire nodi selezionando cosa vogliamo vedere. Nella tendina a sinistra abbiamo anche a disposizione una lista con gli ortologhi di ogni gene; abbiamo anche il dato di confidenza con cui un ortologo è chiamato e questo si basa sulla similarità di sequenza. Ci sono poi sempre nella tendina anche le ontologie e i fenotipi, le patologie a cui è associato e i paper che lo dimostrano; altra informazione delle tendine sono le genetic variation che riporta le varianti del gene, il tipo di variante, le evidenze, le localizzazioni, possibili conseguenze e ulteriori valori che vengono da altri database e altri tool. Abbiamo una heatmap che rappresenta sulle colonne i tessuti e sulle righe i dataset o gli studi da cui proviene l'informazione e più scuro è il colore, più alta è l'espressione di un gene in quel tessuto. Ci sono anche elencati i pathway in cui questo gene è coinvolto. Ci sono anche le varie regolazioni, come ad esempio i microRNA di cui questo gene è target.
Per quanto riguarda i trascritti ce li mostra, ci fornisce il loro nome e il loro ensembl transcript ID, la lunghezza, se sono protein coding o no e dei codici CCDS (Consensus Coding Sequence Set) che vuol dire che queste sequenze sono consenso tra ensembl e altri progetti di annotazione (Havana- in cui l'aggiornamento delle sequenze è manuale-, NCBI, UCSC). I rettangoli che troviamo nella sequenza rappresentano gli esoni mentre le linee rappresentano gli introni; quando gli esoni sono pieni vuol dire che sono codificanti mentre quando sono vuote rappresentano regioni introdotte. Il codice colore inoltre distingue enhancer, promotori, siti di binding e altro; questi sono informativi così a occhio ma se dovessimo andare maggiormente nel dettaglio dovremmo andare a vedere le origini delle informazioni per definire queste regioni più nel dettaglio.
Nella finestra a sinistra, l'indice, c'è una voce chiamata configure region image che ci permette di manipolare un po' la rappresentazione grafica vista finora e anche aggiungere altre informazioni cliccando sulle varianti.
Altra cosa è che la linea blu definisce la sequenza, il trascritto sopra è il forward e quello sotto il reverse.

Inoltre un vantaggio di ensembl è che il manuale è molto completo e, corredato ad ogni titolo troviamo un punto interrogativo con tutte le informazioni che riguardano una sezione di ensembl.
Altro Database di uso molto comune è GTEx (gtexportal.org) che è una raccolta di dati di espressione che sono stati raccolti proprio per questo database (quindi non sono dati che vengono da lavori diversi ma l'espressione dei geni è stata analizzata dal personale di GTEx con protocolli di pipeline tutti standardizzati quindi tutti i dati che troviamo sono stati prodotti e analizzati nello stesso modo e provengono da tessuti umani che sono stati donati a GTEx e che sono stati genotipizzati e sono in generale tessuti sani). L'esempio più semplice è vedere in che maniera il mio gene di interesse è espresso in vari tessuti. Al solito analizziamo la pagina principale che ha sull'estrema sinistra l'area resource overview con i current release quindi la versione momentaneamente caricata del database e subito sotto gli eventi modificati con tutta la documentazione allegata. Semplicemente per cercare un gene possiamo cercarlo nella parte centrale dove richiede il gene ID e arriviamo direttamente alla pagina del gene; anche qui l'ensembl gene ID è segnato con un lungo numero di id che termina con un numero dopo il punto e la parte prima del punto è quella stabile mentre quella dopo il punto può variare con l'aggiunta di esoni. In alto nella pagina del gene troviamo un breve riassunto dello stesso e sotto abbiamo un biomin plot dei valori di espressione con sull'asse y i TPM (transcript per milion) e sulle x vediamo ogni tessuto quindi ci permette di visualizzare il livello di espressione di questo gene in diversi tessuti (può servire se dobbiamo esaminare l'espressione in una linea cellulare diversa e otteniamo un valore di espressione rispetto a quello che otteniamo con la linea cellulare che usiamo di solito e possiamo andare a vedere il livello di espressione di un tessuto rispetto all'altro); inoltre possiamo filtrare su questo plot.
Sotto questo plot troviamo poi l'espressione per esone (più alta è l'espressione e più scuro è il colore), a sinistra troviamo i tessuti e nelle colonne troviamo gli esoni; in alto nella heatmap possiamo anche rispetto a exon vedere l'espressione delle giunzioni in un'altra heatmap e quindi mi si evidenziano le giunzioni tra i due esoni (questa heatmap rappresenta quanto è espressa la giunzione tra i due esoni e quindi quante reads cadono a cavallo tra i due esoni. Questo viene poi riassunto nel gene model che troviamo ancora più in fondo e che ci dimostra ogni campione con il suo ensemble transcript ID, gli esoni per ogni trascritto. Il terzo pannello di questa pagina sono le isoforme che mostrano quale trascritto è più presente nei vari tessuti. Ultimo dettaglio che osserviamo single gene è la regione sui QTLs; in pratica è una raccolta di expression quantitative trait locus che mostra che ci sono delle varianti associate a un diverso livello di espressione. Geni che hanno un gene o l'altro nello snip hanno maggiore o minore espressione in un determinato tessuto.
Tornando all'homepage, se clicchiamo su expression possiamo cercare per gene ID o andare a vedere la multigene query nel caso in cui siamo interessati a una lista di geni più lunga. Nel mapping vediamo lungo le colonne i tessuti e lungo le righe la lista di geni. Se clicchiamo sul singolo gene viene plottata con violin plot l'espressione di quest'ultimo e quindi dove è espresso.
Abbiamo visto cosa fare se sono interessata a un gene o a un gruppo di geni; se sono interessata a un tessuto in particolare torniamo sul menu principale e clicchiamo su top expressed genes e qui arriviamo a una mappa che ci mostra i nostri 50 file più espressi nel tessuto di interesse.
Infine anche per GETex troviamo la documentazione completa in alto e anche alcuni tutorial. Ha una pagina dedicata ai data download cioè ha dei file disponibili come file di testo (sono disponibili le conte, i transcript per milions e tutto ciò che è mostrato nei grafici attraverso questo sito dal momento che tutto è scaricabile).
Ultimo database è GEO (Gene Expression Omnibus), database pubblico e internazionale che serve per raccogliere e conservare dati di esperimenti genomici high-throughput e la comunità dei ricercatori sottomette i propri dati a GEO, i dati vengono accettati e resi pubblici e la comunità ne può usufruire. In questo caso viene condiviso il dato grezzo in FastQ e un summary dei risultati finali che potrebbe essere una lista dei geni differenziali con il loro adj p-value e log2FoldChange. Ovviamente questo non basta ma dobbiamo sottoporre una descrizione dettagliata del disegno sperimentale dei protocolli che abbiamo usato per la parte di laboratorio e di analisi in modo che se qualcuno vuole riprodurre i nostri risultati ha

tutte le informazioni per poterlo fare e ci sono dei regolamenti che salvaguardano questo processo e garantiscono che ad ogni dataset sia allegato un minimo di informazione che renda l'esperimento comprensibile.
In GEO a ogni esperimento viene associato un ID che è quello che si trova nei paper ed è quello che si trova nei paper per poi risalire ai dati. Il contenuto a destra si divide in samples, platforms, series e datasets (platform non lo indaghiamo, è semplicemente la descrizione del sequenziatore che è stato usato o dell'array ed è comune a più esperimenti). In samples abbiamo le condizioni alle quali è stato sottoposto ogni campione e come è stato calcolato e le misure (ogni sample ha il suo assession number che è un GSM) e ogni samples presenta un dato in platform, in series e in organisms ma anche il contatto del ricercatore e quando un dato campione è stato rilasciato; di base in samples non troviamo molte informazioni ma quello che più ci interessa è riportato in "series" che mettono insieme tutti i campioni che sono correlati in un gruppo e fanno parte dello stesso studio e presentano tutte le informazioni sull'esperimento, dai trattamenti ai protocolli ecc. (in fondo abbiamo i dati processati finali che sono i risultati finali ma c'è un link a SRA dove si riferisce al singolo esperimento in cui troviamo le reads in FASTQ- ma è pubblico solo quando ogni esperimento è completato; se dobbiamo scaricare molti dataset, salviamo gli SRA ID e li scarichiamo più rapidamente in pochi passi dalla linea di comando).
Il dataset è una collezione curata dallo staff di GEO che contiene i campioni correlati a livello biologico e supponiamo nella stessa linea cellulare con lo stesso trattamento. Quando apriamo un dataset a destra ci sono alcuni dati tra cui le reference mentre a sinistra è presente la cluster analysis che è la rappresentazione grafica dell'espressione genica in vari campioni che sono all'interno del dataset e ci sono anche disponibili dei files da andare a vedere se siamo interessati a un dato tessuto; questo clustering lo possiamo esaminare selezionando delle parti e andandole a vedere più nel dettaglio. Il messaggio generale di GEO è che i dati analizzati in paper pubblicati vengono resi pubblici e presentati qui e dai paper, con la session number riusciamo a raggiungere la serie di dati in questione e se vogliamo rianalizzarli o semplicemente vedere nel dettaglio i risultati finali.

Gene list analysis

1. Raw reads (FASTQ)
   a. Quality Control (QC)
   b. Trimming - Quality Control
2. Alignment / Quantification

   Preliminary analyses / QC
3. Differential Expression Analysis
4. Functional annotation

Cominciamo rivedendo i passaggi di un esperimento di analisi di geni differenziali tramite analisi del trascrittoma quindi tramite le sequenze grezze in fastQ abbiamo dei controlli di qualità ed eventuali filtri sulle sequenze seguite da un nuovo step di controllo qualità; poi abbiamo allineamento e quantificazione in uno o più step a seconda dell'approccio che vogliamo utilizzare e segue quindi un passaggio di analisi preliminari e controllo qualità che abbiamo fatto in un altro contesto ma non è stato ben inserito in questi passaggi qui. Queste analisi preliminari sono le normalizzazioni che avevamo fatto prima del clustering o prima della PCA e queste ci servono per capire, ancora prima di fare analisi di espressione differenziale, se i nostri campioni potrebbero dare indicazioni di eventuali problemi; questo perché se notiamo dei problemi sulle analisi preliminari possiamo approcciare l'analisi del flusso di espressione differenziale in una maniera più attenta magari tenendo conto di eventuali replicati che non sono andati bene o eliminando un campione che non si comporta come ci aspetteremmo quindi questo serve per focalizzare l'attenzione sui nostri dati e su come sono prima di guardare direttamente i risultati dei geni differenzialmente espressi.
Segue la parte finale di questi passaggi che è la parte di annotazione funzionale che ci permette di individuare in quali processi biologici sono coinvolti i geni differenzialmente espressi.
Ci serve normalizzare per library size: nel caso in cui in un campione vengano sequenziate 20 milioni di reads e in un altro 40, nel secondo campione i geni avranno dei valori di conte doppie rispetto al primo campione; questo falserebbe le nostre analisi perché non ci permetterebbe di concludere che il trascritto sia più espresso nel secondo campione rispetto che nel primo ed è questo il motivo per cui dobbiamo normalizzare per la dimensione della library.
Dobbiamo riprendere i processi effettuati nella sessione precedente in R studio quindi carichiamo le librerie

DESeq2 e airway e i dataset airway e costruiamo il dataset dds, manteniamo i geni che sono espressi almeno 10 volte in ogni reads e separiamo i geni significativi.

```
dds <- DESeqDataSet(airway, design = ~ cell + dex)

keep <- rowSums(counts(dds)) >= 10
dds <- dds[keep,]

dds <- DESeq(dds)
res <- results(dds)

sig = res[which(
          abs(res[,"log2FoldChange"])>1
          & res[,"padj"]<=0.05),]
```

Per la nuova parte di analisi, di annotazione funzionale, dobbiamo richiamare cinque diverse libraries che sono: clusterProfiler, reactome PA, org.Hs.eg.db, enrichplot e GSEABase. Siamo arrivati nelle volte precedenti ad avere il risultato delle nostre analisi di espressione differenziale che consisteva in elenchi di medie di basi, log2FoldChange, lcfSE, stat, p-value e padj che di loro non ci danno tantissime informazioni nel senso che abbiamo una lista di geni e o li conosciamo e quindi possiamo ragionare sull'impatto nel nostro sistema cellulare del farmaco che abbiamo usato.

Un modo sarebbe andare a vedere tutti i geni uno ad uno e un altro sarebbe analizzare la lista di geni con dei metodi e uno di questi è l'arricchimento funzionale utilizzando le gene ontology. Per fare un test di arricchimento usando le gene ontology ci servono una lista di geni (che abbiamo perché è la lista dei nostri geni significativi), le gene ontology e una lista di tutti i geni (l'universo).

Prima di fare l'arricchimento quindi prepariamo queste liste di geni. Il primo comando serve semplicemente per aggiungere alla tabella dei dati significativi anche una colonna con i gene ID e allo stesso modo il secondo comando per i risultati; creiamo quindi una lista di geni (gene.ABS) utilizzando una funzione che, attraverso la library che abbiamo caricato org.Hs.eg.db, ci permetta di avere per ogni ensembl gene ID anche l'entrez ID e il gene symbol questo perché nelle funzioni che usiamo successivamente per fare l'arricchimento serve l'entrez gene ID. Una percentuale dei nostri input gene ID non mappa (non è un problema dei nostri dati, o non trova corrispondenza o magari sono non coding) ma non possiamo fare granché; è importante sapere che da qui in avanti utilizziamo un subset dei nostri dati sperimentali (teniamolo presente nei ragionamenti che faremo sulle funzioni). Siccome facciamo il test per tutti i termini delle GO, prendiamo un termine GO a caso e ci sono in tutti l'universo (tutti i geni umani) magari 30 geni connessi al processo biologico; quello che voglio valutare è se nel mio dataset di geni differenzialmente espressi c'è un arricchimento di geni per questo termine maggiore rispetto alla percentuale rispetto all'universo (ci facciamo questa domanda per tutti i termini dell'universo ed è chiaro che considerando nell'analisi meno geni del totale differenzialmente espresso- perché non mappa- potremmo perdere qualcosa e non vedere magari termini arricchiti come tali però non abbiamo i mezzi per tenerli in considerazione e quindi dobbiamo farci andare bene quello che abbiamo. Nel caso in cui non ci fosse la visualizzazione di un arricchimento confrontato con la GO potremmo fare delle considerazioni aggiuntive come ad esempio andare a guardare i geni singolarmente per vedere se ci suggeriscono qualcosa (magari geni molto sparsi e non riconducibili a un solo pathway). A quel punto si vanno a guardare sottogruppi oppure in base al modello che sto studiando se so che generalmente il modello intacca i geni di un certo pathway vado a vedere se i geni si trovano in quel pathway: non posso dire che sono arricchiti ma posso dire che ci sono in quel pathway e valutare che significato possa avere.

```
#gene list
sig["Hs_ensembl_gene_id"] <- rownames(sig)

resABS <-(sig[,"Hs_ensembl_gene_id"])

gene.ABS <- bitr(resABS, fromType = "ENSEMBL",
      toType = c("ENSEMBL", "SYMBOL","ENTREZID"),
      OrgDb = org.Hs.eg.db)
```

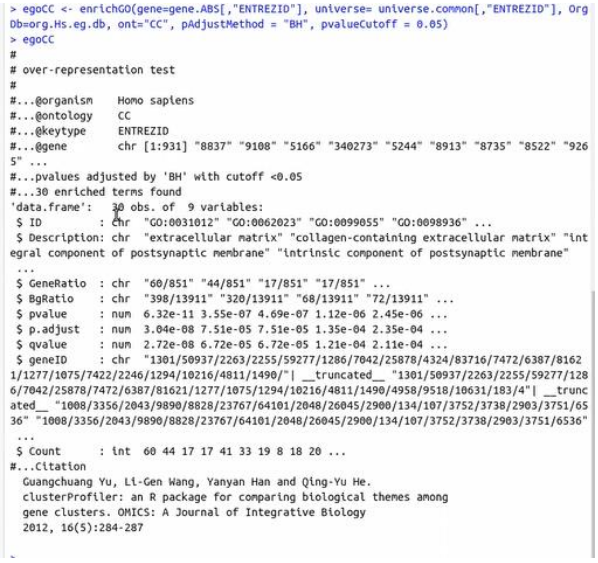
```
#universe
res["Hs_ensembl_gene_id"] <- rownames(res)

universe.common<- bitr(res[,"Hs_ensembl_gene_id"],
fromType = "ENSEMBL",
      toType = c("ENSEMBL", "SYMBOL","ENTREZID"),
      OrgDb = org.Hs.eg.db)
```

Continuiamo compilando le altre due liste di geni: nel nostro caso il nostro universo sono tutti i geni espressi nel genoma di riferimento. Ora che abbiamo le nostre liste pronte possiamo fare il GO enrichment. Ci sono tre aspetti raccolti nelle GO che sono cellular component, biological process e molecular function; noi andiamo a fare un'analisi di arricchimento per ognuno di questi aspetti. Per farlo utilizziamo la funzione enrichGO applicata ai tre differenti aspetti e diamo tutti i parametri necessari.

```
egoCC <- enrichGO(gene     = gene.ABS[,"ENTREZID"],
             universe     = universe.common[,"ENTREZID"],
             OrgDb        = org.Hs.eg.db,
             ont          = "CC",
             pAdjustMethod = "BH",
             pvalueCutoff  = 0.05)

egoBP <- enrichGO(gene      = gene.ABS[,"ENTREZID"],
             universe     = universe.common[,"ENTREZID"],
             OrgDb        = org.Hs.eg.db,
             ont          = "BP",
             pAdjustMethod = "BH",
             pvalueCutoff  = 0.05)

egoMF <- enrichGO(gene        = gene.ABS[,"ENTREZID"],
             universe     = universe.common[,"ENTREZID"],
             OrgDb        = org.Hs.eg.db,
             ont          = "MF",
             pAdjustMethod = "BH",
             pvalueCutoff  = 0.05)
```

Come geni dobbiamo utilizzare gli entrez ID per tutti gli arricchimenti, poi dobbiamo specificare l'universo, la libreria, l'ontologia (CC=cellular component, BP biological process e MF molecular function) e diamo la soglia del padj.

Possiamo quindi accedere ai dati di over-representation test che definisce varie cose di cui le prime comuni alle rappresentazioni precedenti e che sono i parametri che noi abbiamo fornito quindi organismo, ontologia, l'entrez ID, il gene e il cromosoma, l'ID, il p-value adjusted e una parte di data-frame in cui definisce che ci sono 30 termini arricchiti identificati con l'Id e possiamo anche ricevere in output la lista dei geni arricchiti; per ciascun termine ci fornisce una descrizione grazie al quale riusciamo a capire di cosa stiamo parlando. Fornisce anche informazioni di gene ratio che è la lista e il background ratio che è il rapporto per ogni termine dei geni su tutto l'universo.



```
> egoCC <- enrichGO(gene=gene.ABS[,"ENTREZID"], universe= universe.common[,"ENTREZID"], Org
Db=org.Hs.eg.db, ont="CC", pAdjustMethod = "BH", pvalueCutoff = 0.05)
> egoCC
#
# over-representation test
#
#...@organism    Homo sapiens
#...@ontology    CC
#...@keytype     ENTREZID
#...@gene        chr [1:931] "8837" "9108" "5166" "340273" "5244" "8913" "8735" "8522" "926
5" ...
#...pvalues adjusted by 'BH' with cutoff <0.05
#...30 enriched terms found
'data.frame':   30 obs. of  9 variables:
 $ ID         : chr  "GO:0031012" "GO:0062023" "GO:0099055" "GO:0098936" ...
 $ Description: chr  "extracellular matrix" "collagen-containing extracellular matrix" "int
egral component of postsynaptic membrane" "intrinsic component of postsynaptic membrane"
 ...
 $ GeneRatio  : chr  "60/851" "44/851" "17/851" "17/851" ...
 $ BgRatio    : chr  "398/13911" "320/13911" "68/13911" "72/13911" ...
 $ pvalue     : num  6.32e-11 3.55e-07 4.69e-07 1.12e-06 2.45e-06 ...
 $ p.adjust   : num  3.04e-08 7.51e-05 7.51e-05 1.35e-04 2.35e-04 ...
 $ qvalue     : num  2.72e-08 6.72e-05 6.72e-05 1.21e-04 2.11e-04 ...
 $ geneID     : chr  "1301/50937/2263/2255/59277/1286/7042/25878/4324/83716/7472/6387/8162
1/1277/1075/7422/2246/1294/10216/4811/1490/"| __truncated__ "1301/50937/2263/2255/59277/128
6/7042/25878/7472/6387/81621/1277/1075/1294/10216/4811/1490/4958/9518/10631/183/4"| __trunc
ated__ "1008/3356/2043/9890/8828/23767/64101/2048/26045/2900/134/107/3752/3738/2903/3751/65
36" "1008/3356/2043/9890/8828/23767/64101/2048/26045/2900/134/107/3752/3738/2903/3751/6536"
 ...
 $ Count      : int  60 44 17 17 41 33 19 8 18 20 ...
#...Citation
  Guangchuang Yu, Li-Gen Wang, Yanyan Han and Qing-Yu He.
  clusterProfiler: an R package for comparing biological themes among
  gene clusters. OMICS: A Journal of Integrative Biology
  2012, 16(5):284-287
```

Valutiamo gli stessi valori per tutte le ontologie (semplicemente modificando la voce ontology e ogni volta valutiamo l'arricchimento. Nel caso del biological process troviamo addirittura 566 oggetti mentre nel caso del molecular function ne troviamo 38.

Finché i termini sono 38 e 30 li possiamo leggere e vedere in quali componenti cellulari i nostri geni differenzialmente espressi sono coinvolti. I dati ricavati da questa gene ontology possono essere utili nel caso in cui non sappiamo in che comparto cellulare agisca il nostro farmaco e dove avvengano le conseguenze.

Nel caso della molecular function ci dice invece che ruolo svolgano i geni che stiamo esaminando ed è più facile capire quali siano le conseguenze del trattamento farmacologico ma ancora non abbiamo idea dei processi di cui questi geni fanno parte.

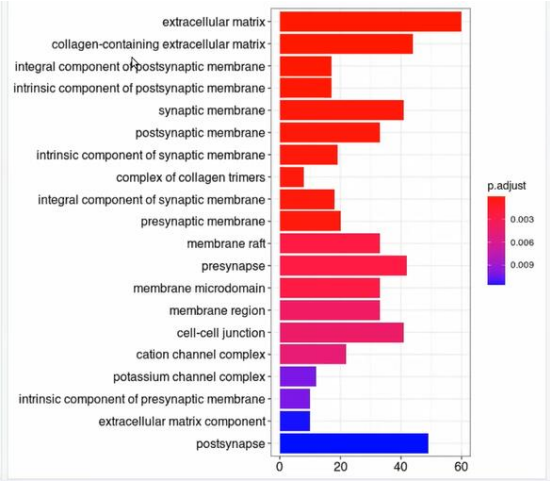
Nel caso dei biological process i dati sono veramente tanti e, mentre gli altri si possono leggere al volo, questi sono difficile da interpretare in lista. Questo risultato è da gestire in maniera diversa e lo faremo a breve.

Abbiamo visto i tre ambiti della gene ontology e ora vediamo i pathway (reactome pathway database). Per farlo usiamo il tool online reactome, un database che raccoglie i geni raccolti in determinati pathway catalogati per numero. Troviamo online questa mappa con tutti i pathway e zoomiamo su quello di interesse; cliccando su un ramo e scorrendo la lista di dettagli forniti troviamo anche i geni coinvolti in un dato path (c'è un minimo di descrizione, il compartimento cellulare interessato, le pubblicazioni che portano evidenze per questo pathway, ci sono poi le molecole coinvolte, le strutture di queste molecole, i livelli di espressione e altri dati; ogni pathway ha un ID preciso). Rispetto alle GO, i pathways sono un altro tipo di database per cui possiamo integrare entrambe le informazioni per avere un quadro delle informazioni più chiaro possibile. Possiamo fare quello che faremmo su Reactome online anche su R lanciando l'arricchimento da Reactome semplicemente con la funzione enrichPathway (egoReactome <- enrichPathway (gene=gene.ABSS [,"ENTREZID], pvalueCutoff=0.05, readable=T)). Questo perché dati che stiamo analizzando li abbiamo già su R ed è comodo, con il tool online dovremmo darlo in input; inoltre se non sto lavorando in umano è difficile che io riesca a fare la parte di annotazione funzionale perché anche per gli organismi, se non per il topo, non ci sono i pathway, ci sono solo alcune gene ontology di alcuni

organismi; quindi se il mio organismo modello è zebrafish, una volta che ho i geni differenziali in zebrafish passo agli ortologhi umani. È da notare che se non c'è corrispondenza 1:1 tra gli ortologhi non cambiamo il numero dei geni (se in zebrafish ho due proteine che in umano hanno lo stesso ortologo devo fare attenzione all'annotazione funzionale per non raddoppiare l'espressione in maniera errata quindi bisogna stare attenti e selezionare i geni in modo che siano corrispondenti 1:1 altrimenti ottengo informazioni fuorvianti nell'arricchimento.

Possiamo contemporaneamente valutare l'arricchimento valutando da R oppure tramite Reactome online ma dobbiamo tenere chiaramente conto che la stringenza magari che la funzione su R applica non è la stessa che applica la pagina Web.

Fatta questa analisi vediamo che non abbiamo arricchimento nei pathway mentre le gene ontology ci suggerivano tanti arricchimenti; questo succede perché sono approcci diversi e i pathway sono annotati con i geni coinvolti ma hanno una struttura diversa da quella delle GO e molto più generale.



Possiamo mostrare i risultati dell'arricchimento delle GO con dei banali barplot; ottenuto il risultato delle GO o lo mostriamo in tabella o creiamo una rappresentazione grafica. Nel barplot in questione il colore delle barre corrisponde al valore di adjusted p-value e la lunghezza delle stesse all'arricchimento (quanti geni differenziali sono presenti annotati a un termine). Il comando che utilizziamo è barplot (egoCC, showCategory=20). E allo stesso modo poi possiamo fare per gli altri geni della gene ontology.

Esiste un set della gene ontology che si chiama slim GO in cui sono contenute gene ontology tagliate molto vicino alla radice quindi tagliamo i termini dettagliati e teniamo solo i generali e questo può essere utile per magari riassumere dei termini più specifici quando ne abbiamo tanti come in questo caso.

```
egoBP2 <-setReadable(egoBP,OrgDb = org.Hs.eg.db)

write.table(file="BP_enrichment_airway.txt",
as.data.frame(egoBP2), sep="\t",row.names=F)
```

Andiamo poi a scrivere questa tabella in un file in modo da poterla avere se ci serve.

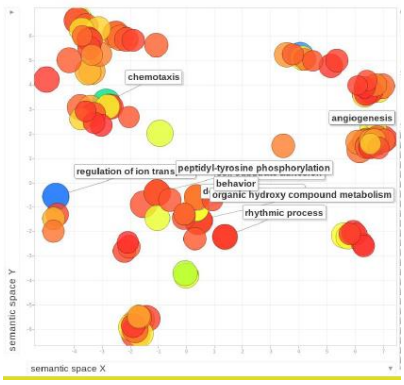
Questa tabella avrà le colonne separate dai tab: nella prima colonna sarà contenuto l'ID dei termini GO, poi la descrizione, quindi il gene ratio della nostra lista e il ratio sul background per ogni termine e il p-value.

Nel caso in cui vogliamo la tabella possiamo formattare questa o fare di plot di tipo diverso.

In alternativa c'è un altro tool online che si chiama REVIGO (REduce+VIsualize Gene Ontology) che permette di rappresentare in maniera in maniera un po' più compatta i termini della GO.

```
BP_rev <- as.data.frame(egoBP$ID)
BP_rev['p.adjust'] = egoBP$p.adjust

head(BP_rev)

write.table(file="airway_egoBP_rev",BP_rev,sep="\t",row.names=F,
col.names=F, quote=F)
```

Proviamo a caricare le nostre GO su REVIGO per vedere se riusciamo a trovare un modo per riassumere tutti i termini che abbiamo trovato arricchiti per i biological process; per fare questo dobbiamo scrivere in un file i GO ID e gli adjusted p-value (BP_rev) e poi scriviamo una tabella con questi files. REVIGO vuole che incolliamo questa tabella nella sezione di input e poi possiamo far partire l'analisi.



L'output è quello che vediamo nell'immagine a fianco con il colore che rappresenta l'adjusted p-value e vediamo chiaramente che alcuni geni risultano annotati nel grafico connessi ai diversi pathway.

Il raggruppamento dei termini per similarità ci aiuta a capire meglio cosa stiamo guardando e a riassumere i pathway principali (non vale troppo come rappresentazione ma può valere come rappresentazione intermedia dei GO che riteniamo importanti e ci alleghiamo la tabella completa o altre rappresentazioni).

A questo punto ci possiamo occupare della GSEA (Gene Set Enrichment Analysis) che funziona in modo un po' diverso rispetto al precedente. In input per la GSEA dobbiamo fornire una lista di geni che comprenda tutti i geni espressi ma questi geni sono ordinati. Per ordinare i geni dobbiamo fare un ranking dei nostri geni in base, ad esempio, al Log2FoldChange e quindi andiamo a vedere se c'è un arricchimento nei termini della GO alle estremità della nostra lista ordinata di geni (quindi se c'è un arricchimento dei geni che cambiano di più in aumento o in decremento).

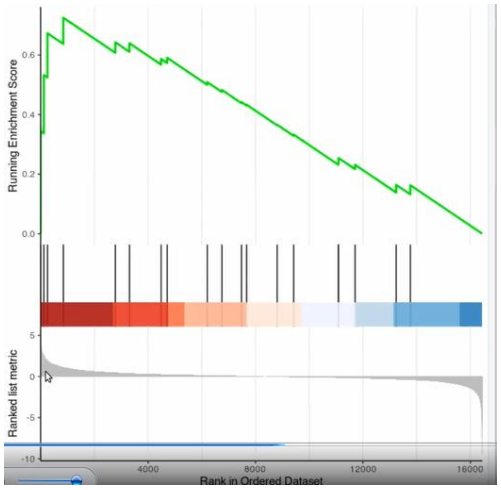
```
x.gsea<-res[order(as.numeric(as.character(res[,"log2FoldChange"]
)),decreasing=T),"log2FoldChange"]

   names(x.gsea)<-universe.common[order(as.numeric(as.character(
res[,"log2FoldChange"])),decreasing=T),"ENTREZID"]

x.gsea<-x.gsea[-which(is.na(names(x.gsea)))]
```

Il nostro oggetto da dare in input viene dall'oggetto res ordinato per i valori di log2FoldChange in maniera decrescente; andiamo ad aggiungere gli ID e poi togliamo dalla lista i geni che non sono stati mappati e quindi si trovano senza ID.

A questo punto possiamo procedere con l'arricchimento da effettuare con la funzione gseGO e i parametri che settiamo, così come le funzioni, sono analoghe alle ontologie solo che alla voce "ontologie" diamo il parametro "ALL" perché facciamo l'analisi per tutte e tre le GO insieme.

```
gseaALL <- gseGO(gene          = x.gsea,
                 OrgDb         = org.Hs.eg.db,
                 ont           = "ALL",
                 pAdjustMethod = "BH",
                 pvalueCutoff  = 0.05,
                 keyType = "ENTREZID")
```

Vediamo che secondo la GSEA non abbiamo nessun arricchimento;



se avessimo visto arricchimento avremmo creato un plot con la funzione gseaplot2 (gseaALL, geneSetID = 1) e quello che avremmo ottenuto è un grafico come quello nell'immagine in cui viene mostrata la posizione dell'arricchimento (nel senso di overespressione o viceversa) dal momento che nell'asse delle x visualizziamo il rank basato sul fold change con i geni arricchiti illustrati lungo la barra colorata e rappresentati come stanghette e sulla y invece l'enrichment score calcolato sul set di geni. L'arricchimento è in questo caso è sui geni upregolati e la diminuzione della linea verde è dovuta all'assenza di quei geni nei gene set. Difficilmente abbiamo un alto valore di enrichment score nella parte centrale visto che al centro sono presenti i geni che non cambiano e con fold change pari a zero (e non avrebbe senso).

Dobbiamo ricordare che con questo plot non grafichiamo l'intero set ma dei subset ad esempio per rappresentare l'impatto dell'arricchimento su un pathway graficheremo i geni di quel pathway.

Per terminare vediamo enrichr che è un sito che ci permette di vedere l'arricchimento per delle liste di geni. Possiamo registrarci e fare il log-in per salvare le nostre analisi altrimenti gira senza fare accesso e mi permette di salvare i dati; sono disponibili i dati per umano ma anche per insetti, pesci e lievito che è molto comodo. Per preparare il file di input su R costruiamo una tabella che riporti i gene symbol dei nostri geni significativi (a cui abbiamo aggiunto il gene symbol mentre avevamo solo gli ensembl ID e rimuoviamo le informazioni aggiuntive e i tab e le virgolette. Salvato il file con la lista possiamo caricarlo su enrichr (oppure copiare e incollare la lista) e questo ci fornirà diversi output: in alto sotto il titolo abbiamo varie tabelle che riguardano database diversi (uno per la trascrizione, uno per i pathways, uno per trattamento o malattie, uno per i tipi cellulari e tanto altro). Andiamo a osservare i dati basati su KEGG che è un altro database di pathways (come Reactome) che ci mostra immediatamente un output grafico che in maniera standard è ordinato per p-value ma ci sono alti metodi (come il combined score che è un tipo di ranking basato sia sul p-value che su altro) e possiamo chiaramente vedere la significatività basandoci sul colore; possiamo trovare i risultati in forma di tabella. Questo quindi è un metodo alternativo a quello da noi visualizzato, o anche complementare, a cui possiamo applicare le stesse considerazioni fatte precedentemente. I risultati non sono completamente sovrapponibili perché c'è da considerare il diverso set di GO che usano le funzioni che utilizziamo e inoltre bisogna vedere come viene fatta la correzione del

multiple testing. Questo è un modo più immediato per avere i risultati ma bisogna scavare un po' di più per capire come funziona e tutti metodi utilizzati perché siamo noi a dettarli.

Quindi l'annotazione funzionale, in sintesi, può avvenire tramite diversi approcci e questi approcci possono anche essere svolti in parallelo. Questi approcci sono: GO enrichment, pathway enrichment e GSEA.

Abbiamo visto la tree map di REVIGO, una formulazione grafica del legame tra la significatività e una certa annotazione; la dimensione dei rettangoli dipende dal p-value o dalla frequenza del termine nell'intero database e non nel dataset perché non è un'informazione che noi forniamo tra i dati di input a REVIGO.

Systems biology

Vogliamo vedere un esempio di programma che ci permetta di visualizzare un protein-protein interaction network. Utilizziamo ancora la lista di geni differenzialmente espressi che abbiamo trovato le volte scorse. (Quindi installiamo le due librerie e importiamo il dataset, costruiamo il dds, manteniamo i geni la cui espressione è significativa, applichiamo DESeq e ricaviamo i risultati e i geni significativi.) Prepariamo i file di input per visualizzare il network cioè creiamo un dataframe in cui abbiamo i nostri geni differenzialmente espressi e in cui mettiamo anche l'ID che avevamo identificato (non solo il gene ID che è il nostro row name); facciamo il merge utilizzando l'ensembl gene ID come colonna su cui fare il merge (quindi viene fuori lo stesso sig a cui però aggiungiamo il gene symbol e l'entrez. Possiamo quindi scrivere questa tabella in un file di output con la funzione write.table.

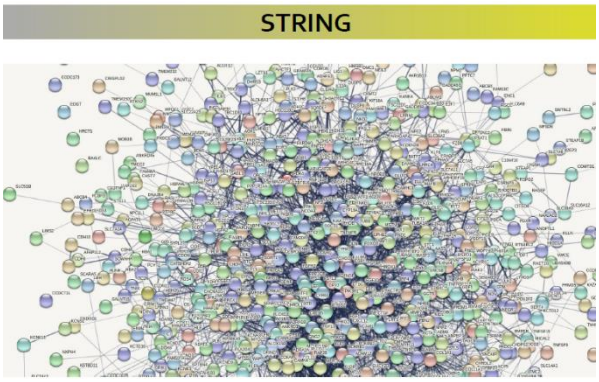
```
tocys <- merge(as.data.frame(sig),gene.ABS,
by.x='Hs_ensembl_gene_id',by.y='ENSEMBL', all=TRUE)


write.table(tocys, file="airway_expression.tsv", sep="\t",
col.names = T, row.names = F, quote = F)
```

Preparati i files andiamo quindi a scoprire un database di protein-protein interaction che si chiama STRING. String ha una sezione che si chiama download per scaricare dati completi: tutte le protein protein interaction identificate in un organismo e tutti gli organismi sono selezionabili. Nell'area "search" troviamo un indice in cui sono elencate le possibilità di ricerca cioè si possono cercare le proteine per nome, per sequenza, si può fare la ricerca per multiple proteins e multiple sequences. Nel nostro caso siamo interessati alla ricerca multiple proteins perché dobbiamo fare la ricerca su tutti gli ID dei geni differenzialmente espressi; con questa selezione possiamo andare a vedere l'interazione proteina-proteina tra tutti i nostri geni differenzialmente espressi. Selezioniamo l'organismo e carichiamo il file di tutti gli ensembl gene ID (con la linea di comando tramite cut selezioniamo una colonna del nostro file di ensembl oppure selezioniamo tramite R una colonna da inserire in un altro file). Caricato il file, con il nostro enseml gene ID le proteine vengono identificate; otteniamo infine in output il network delle proteine in formato TSV (Tabular Separated Values) che è un formato di file che nella prima colonna presenta una proteina, nella seconda colonna la seconda proteina e la riga rappresenta l'interazione tra le due.

RSPO1	LGR5	4441875	4435721	9606.ENSP00000348944	9606.ENSP00000266674	0	0	0	0	0	0.846	0.900	0.877	0.997
PNPLA2	LIPE	4440813	4433889	9606.ENSP00000337701	9606.ENSP00000244289	0	0	0	0	0.087	0.186	0.900	0.952	0.996
ABHD5	PNPLA2	4447673	4440813	9606.ENSP00000390849	9606.ENSP00000337701	0	0	0	0	0.064	0.472	0.800	0.972	0.996
GCH1	GCHFR	4449007	4434783	9606.ENSP00000419045	9606.ENSP00000260447	0	0	0	0	0.062	0.831	0.900	0.797	0.996
HBA1	HBA2	4439524	4434183	9606.ENSP00000322421	9606.ENSP00000251595	0	0	0	0.985	0.771	0.800	0.900	0.538	0.995
TUBB	TUBA1A	4440915	4437720	9606.ENSP00000339001	9606.ENSP00000301071	0	0	0	0.920	0.227	0.948	0.900	0.591	0.995
ING2	SAP30	4438295	4437294	9606.ENSP00000307183	9606.ENSP00000296504	0	0	0	0	0.062	0.823	0.900	0.730	0.994
CDKN1A	NOV	4447168	4434710	9606.ENSP00000384849	9606.ENSP00000259526	0	0	0	0	0	0.993	0	0.111	0.993
PLXNA4	SEMA3A	4442296	4435593	9606.ENSP00000352882	9606.ENSP00000265362	0	0	0	0	0	0.365	0.900	0.905	0.993
CYFIP2	WASF3	4450126	4440633	9606.ENSP00000444645	9606.ENSP00000335055	0	0	0	0	0.083	0.813	0.900	0.642	0.993
LEP	PPARG	4438805	4436745	9606.ENSP00000312652	9606.ENSP00000287820	0	0	0	0	0.061	0	0.900	0.915	0.991
PPARG	FABP4	4436745	4434459	9606.ENSP00000287820	9606.ENSP00000256104	0	0	0	0	0.087	0.064	0.900	0.915	0.991
JUN	BATF3	4443699	4433855	9606.ENSP00000360266	9606.ENSP00000243440	0	0	0	0	0.064	0.780	0.900	0.621	0.991
MSTN	FST	4434810	4434510	9606.ENSP00000260950	9606.ENSP00000256759	0	0	0	0	0.050	0.845	0	0.940	0.990
TUBB	TUBA4A	4440915	4434056	9606.ENSP00000339001	9606.ENSP00000248437	0	0	0	0.920	0.189	0.881	0.900	0.679	0.990
VEGFA	THBS1	4451310	4434771	9606.ENSP00000478570	9606.ENSP00000260356	0	0	0	0	0	0.379	0.900	0.852	0.990
LGR4	RSPO2	4445176	4436239	9606.ENSP00000368516	9606.ENSP00000276659	0	0	0	0	0	0.477	0.900	0.811	0.989
DSC3	PKP2	4442377	4432648	9606.ENSP00000353608	9606.ENSP00000070846	0	0	0	0	0.062	0.169	0.900	0.878	0.989

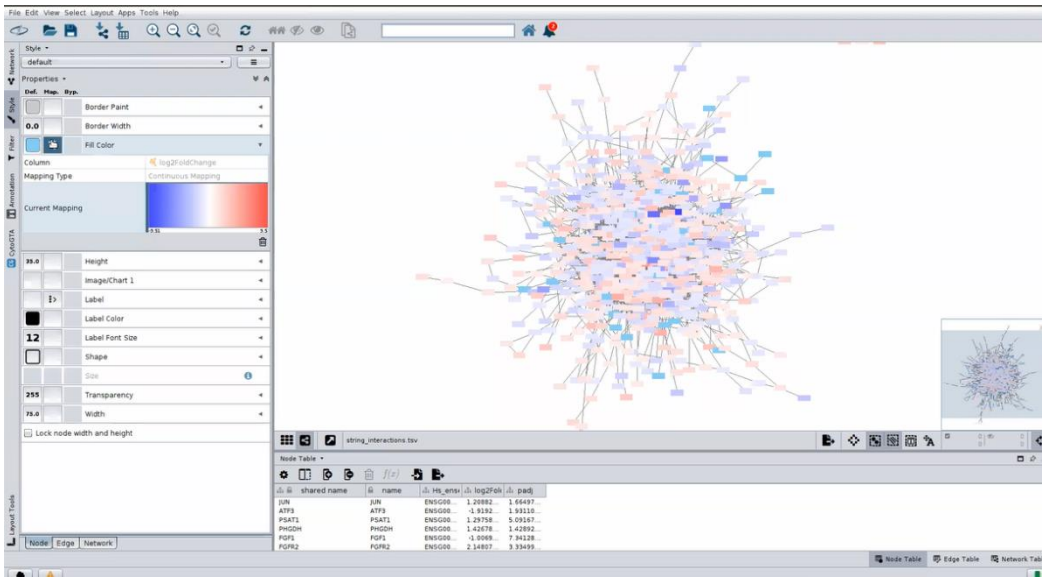
Questo formato di file è quello di cui abbiamo bisogno per, successivamente, caricare i files su cytoscape. Questo file è già la rappresentazione di un network, String ci ha permesso di avere informazioni sulle interazioni tra i nostri geni di interesse e da lì scarichiamo il file che contiene le informazioni di associazione tra le nostre proteine ma non solo, tutti i geni con cui interagisce sono presenti nella lista.

A questo punto si può accedere a cytoscape e caricare questa tabella, string interaction; per caricare il file clicchiamo su file-> import-> network from file e selezionare la nostra tabella.

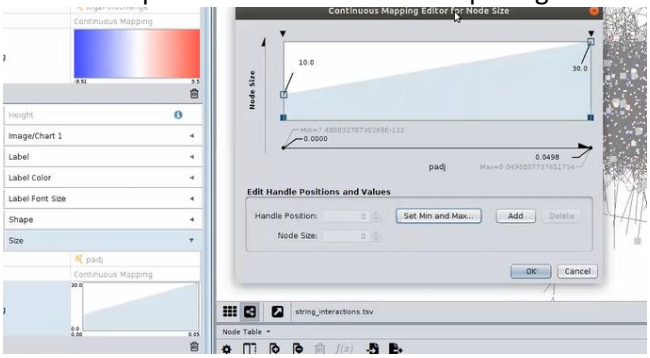


Tramite cytoscape possiamo modificare la visualizzazione del network aggiungendo l'informazione sui geni differenzialmente espressi. Abbiamo un file di input che è il nostro network preso da string e carichiamo i nostri geni di interesse; facciamo anche la ricerca di una proteina singola perché se io sono interessata a quella proteina e alle proteine con cui interagisce perché devo fare coimmunoprecipitazione e mi serve un controllo positivo (lo voglio coimmunoprecipitare, mi serve un controllo positivo per sapere con cosa sono sicura che precipiterà e lo trovo su string cercando la proteina singola). Invece quando vogliamo visualizzare una network e aggiungere delle altre informazioni dobbiamo integrare con cytoscape (mentre se sono banalmente questioni di visualizzazione semplice possiamo modificare l'immagine su string come ad esempio possiamo modificare i colori delle connessioni tra proteine in base a che tipo di interazione intercorra).
Passiamo a cytoscape e tentiamo di aggiungere informazioni sull'espressione dei nostri geni e in particolare se sono upregolati o downregolati. Per fare questo importiamo il file di input che abbiamo scritto prima su R quindi i due file di input adesso sono string interactions (che abbiamo preso da string e riporta tutte le interazioni proteina-proteina della lista di geni differenzialmente espressi) e il file con le informazioni sull'espressione dei nostri geni. Inportiamo quindi la tabella "to a network collection" perché in questo modo è resa accessibile alla nostra network (selezioniamo solo le colonne che ci interessano che sono l'ensembl gene ID (attributo), il gene symbol (la chiave perché il gene symbol è quello utilizzato nella network per identificare le nostre proteine), l'adjusted p-value e il Log2FoldChange (informazioni che vogliamo che i nostri geni portino con loro). L'immagine del network non cambia ma dovrebbe comparire un'anteprima di tabella sotto il network. Ora possiamo modificare il nostro network selezionando il pulsante "style" sulla barra verticale sinistra: ci si apre un menu con molte possibilità e in basso al pannello leggiamo cosa si modifica utilizzando un dato pannello (ad esempio i nodi). Per prima cosa andiamo a modificare il colore: nella colonna mapping selezioniamo il quadratino "fill color" e ci si apre un sottomenu con i parametri column (in cui selezioniamo il log2foldchange) e mapping type (in cui selezioniamo continuous mapping perché vogliamo che il colore cambi in maniera continua lungo il cambiamento dei valori invece che in maniera discreta); selezionando il continuous mapping si apre la finestrella del colore che selezioniamo e cambiamo la palette selezionando quella che preferiamo.

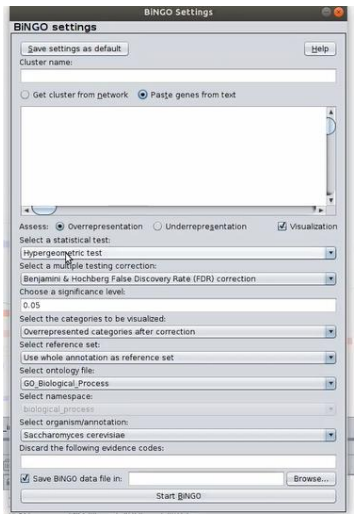
La seconda modifica che facciamo ai nodi è una modifica della dimensione; clicchiamo "lock node width and height" e in questo modo possiamo selezionare la dimensione dei nostri nodi (nel parametro colonna selezioniamo l'adjusted p-value- quindi la dimensione cambia in base alla significatività- e per il mapping type di nuovo selezioniamo continuous

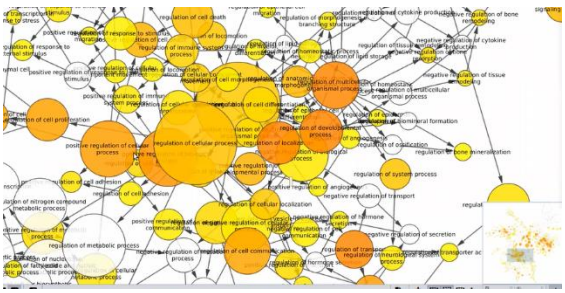


che ci permette di selezionare il tipo di grafico che preferiamo).



Il grafico presenta sull'asse y la dimensione del nodo e sull'asse x tutti i nostri valori di p adjusted. Noi vogliamo che per una questione visiva i geni più significativi siano rappresentati da nodi più grandi quindi minore è l'adjusted p-value maggiore è la dimensione del nodo; questo è un andamento opposto a quello del grafico quindi dobbiamo modificare il grafico alzando al massimo la dimensione del nodo (trascinando verso l'alto il quadratino fino al 30) in corrispondenza del p-value minimo e a p-value massimo portiamo la dimensione del nodo a 10.
Ci sono altre caratteristiche che possiamo modificare come colore e dimensione dell'etichetta.
Un altro esempio di modifica comuneè la modifica degli archi, gli edges, che selezioniamo in basso a sinistra nel pannello. Il pannello degli edges è simile a quella dei nodi e andiamo a modificare allo stesso modo il colore e la dimensione degli archi a seconda dello score dell'evidenza sperimentale con cui è stata dimostrata l'interazione. Selezionando il quadratino "map" del colore ci si apre una tabella analoga a quella del colore nei nodi e al parametro "column" andiamo a selezionare, e quindi renderlo dipendente da, "experimentally determined" che è uno score che rappresenta l'affinità tra le due proteine ed è uno score generale per tutte le interazioni del network (siccome string prende le informazioni da diversi tipi di esperimenti e da diversi database questo è uno score generale); assegno quindi, magari, un colore più scuro per gli archi che rappresentano l'interazione più forte tra due proteine perché hanno l'affinità più alta. Per quanto riguarda il mapping type selezioniamo nuovamente il mapping continuo e scegliamo il colore in modo che graficamente sia più affine a quello che voglio chiarire possibile. In aggiunta cambiamo anche lo spessore degli archi sempre in funzione di questo valore selezionando width alla fine del menu e rendendolo dipendente dalle determinazioni sperimentali, impostando il mapping continuo e mantenendo il grafico che più si confà ai nostri bisogni.
L'ultimo passaggio è vedere un'analisi ulteriore che cytoscape fornisce che è un'analisi del GO enrichment. Mentre l'altra volta abbiamo visualizzato il barplot che abbiamo plottato in R e poi abbiamo visto le rappresentazioni di REVIGO, questa è un'altra possibilità che ci permette di unire l'arricchimento funzionale che abbiamo visto l'altra volta e l'utilizzo di cytoscape per visualizzare una network. Per fare questo lasciamo il tab style e torniamo sul tab network in alto a sinistra, clicchiamo sul menu apps nella parte più alta della schermata del programma e quindi "BINGO". Ci si apre una finestra e selezioniamo i parametri della nostra analisi. Andiamo a incollare nella finestra i nostri geni e selezioniamo come tipo di analisi il test di over representation utilizzando l'hypergeometric test e come correzione manteniamo il Benjamini; piazziamo la soglia di significatività a 0.05 e manteniamo gli altri parametri (a parte l'organismo che dobbiamo selezionare). BINGO quindi farà analisi di enrichment per le GO biological process e dare una rappresentazione dei risultati sotto forma di network. Nell'output abbiamo una tabella che contiene tutte le informazioni usate in questo tipo di analisi.





Oltre alla tabella viene fornito un output grafico.
In questa rappresentazione la dimensione dei nodi è proporzionale al numero dei geni differenzialmente espressi che sono associati al termine del GO. Il secondo punto è la colorazione dei nodi: i nodi vengono colorati di giallo al raggiungimento della soglia settata; al diminuire del p-value il colore del nodo si sposta verso l'arancione quindi più scuro è l'arancione e più significativo è l'arricchimento del termine; i bianchi non sono arricchiti. Possiamo cambiare layout del subset verso il gerarchico.