A set of notes by Charlie **Spring 2017**

Notes on an ‘Genomic Data Science with Galaxy’

Second Module of the Coursera Genomic Data Science Specialization from John Hopkins University

Week One: Introduction

The Galaxy Project (http://galaxyproject.org/) provides (free) user friendly software for producing reproducible genomic pipelines for data analysis. Galaxy is a web-based platform for performing data intensive science. It is particularly suited for genomics, and tools for many different types of genomic data analysis have been integrated in Galaxy. Galaxy is motivated by the need for reproducibility - fueled by the data intensive nature of modern day biology which are heavily dependent on complex statistical methods. Reproducibility of genomic studies is a ‘crisis’. Vasilevsky et al. (2014) show that a tiny fraction of ‘high impact’ papers make clear the exact tools and software they used to conduct research. Nekrutenko and Taylor (2012) show how not only the tools, but also the versions of the tools can create different outcomes. Note: reproducibility is not correctness! It means that an analysis is described in sufficient detail that it can be reproduced by another person in another environment. However, most published analyses are not reproducible - missing software, versions, parameters (etc). Recommendations for research:

1. Accept that computation is critical.
2. Always provide access to raw primary data.
3. Record all auxiliary version of all datasets.
4. Store the exact versions of all software used, and archieve it.
5. Record all parametres, even default ones.

Galaxy is one solution: an analysis environment where all the details and provenance are automatically documented. It’s also opensource and can be extended as required - it’s also extensible for sharing tools, datatypes, workflows, etc. There are four main features to it:

1. Analysis tool - the primary unit.
2. Analysis environment - takes the tools and gives them a uniform UI.
3. Workflow system - for complex multi-step analysis which can then be re-run in one shot.
4. Importantly, it has pervasive sharing and documentation with integrated analysis for collaboration.
5. It also provides a platform for visualisation.

Galaxy offers free accounts, with limitations on data-storage and the number of jobs which can be ran. The two main alternatives are: i.) locally, ii.) on the cloud (most easily on AWS).

Week Two: Galaxy 101

We’re now going to look at human chromozone 22: which coding exons have the largest number of repetitive elements overlapping them? First, we need to get data into Galaxy, then identify which exons have repeats, count repeats, then save and download exons with the most repeat. **Refresher:** The term exon refers to both the DNA sequence within a gene and to the corresponding sequence in RNA transcripts. We’re going to undertake our analysis from https://usegalaxy.org. The ‘Get Data’ tab connects directly to a large range of genomic databases for analysis directly. In our example, we’re going to use the UCSC Main table browser - a database associated with their genome browser - hg19, chr22, output format: BED (browser extensible data, send output data to Galaxy). Galaxy fetches data in the background: while we’re waiting we can continue to use Galaxy. A job is queued (grey), in progress (yellow), failed (red) or completed (green). Clicking gives the preview, and the eye format shows us the data. The bed format gives us the chromozone, position, name, score, strand, etc. To count overlaps, use the 'join' tool in the 'operate on genomic intervals'. When pairs are found, it turns green to show the job is completed. The 'pencil' icon allows us to edit attributes (e.g. field names). After our join is finished, we'll have 6 fields from the exon field, and 6 from the repeat files, joined side by side where there's overlap. To count the number of repeats: 'grouping tool' - join, subtract and group - Group. We then want to group by column 4 - our exon variable. We then want to 'add new operation': 'count' on column 4, and execute. We can then insert operation, and count. Note: for every task and dataset, Galaxy is keeping the provenance of each operation and storing it for reproducibility. The 're-run' button allows us to slightly re-parameterize the problem and run it in a slightly different way. To get data for output: we can join on names on the group, as before. Summary:

- Interactive analysis in Galaxy is performed using tools to operate on datasets.
- Datasets are **immutable**: running tools always creates one or more new datasets.
- Datasets are available through 'history' - which gives complete provenance.

Galaxy work-flows try and take abstract analysis and allow it to be run on other datasets. We can create work-flows from scratch using the work-flow editor, or perform them by example - within the analysis directly, and then extract the work-flow. For example, we can make saved histories our current histories using the history tab on the right of the interface. All of the steps which are undertaken previously can be brought forward and ran all at once sequentially for different datasets. You also have datasets which can be marked specifically for output. 'Post-step' actions can rename datasets, change datatype, assign columns, email notification, etc.

One of the main strengths of Galaxy is the ability to share and collaborate with other users. Nearly all Galaxy entities can be tagged (short structured metadata) or annotated (free form description). Above the datasets there are the 'tag' and 'speech bubble' icon. The tags can be used for searching and be used to find specific histories. Galaxy allows us to share with a specific user (via email or link), or to publish to everyone publicly. There's also a tab of all published histories of all users, etc. The community can actually rate the quality of the shared objects. You can also publish and disable links very easily. In addition to sharing and publishing, there are also 'Galaxy Page's: text documents which permit embedded galaxy objects (such as workflows). Pages also have formatting for headers , etc. The page can then be viewed with objects/histories/etc embedded - similar to an iPy notebook?

Week 3: Working with Sequence Data

Quality control for sequence data: Trial with Illumina iDEA datasets: BT20 paired-end RNA-seq subsamples - search for it through the Data Library. Click on the eye to preview the first 1mb. We can see that this data format is based on records of 4 lines each, where each record begins with an @identifier, followed by sequence identifier and then an encoded quality score. Keys tell us how to map characters e.g. Sanger encoding. FastQC can provide us with information and quality metrics about the data. To access it: NGS: QC and manipulation: FastQC Read Quality reports which gives us two datasets - one is a report which is a `.html` page with summary information and plots (e.g. quality scores). We can also use these tools to trim specific sequences, and only look for regions of certain lengths which have quality scores above a certain level. What tools do you need to use? It depends on your downstream analysis. Summarize: many factors which affect the quality of DNA sequencing data. FASTQC is one tool which allows us to evaluate these metrics. Galaxy provides FASTQC manipulation options to 'recover' even low quality data to salvage for analysis.

We now move to **ChIP-Sequence Analysis with MACS** - map locations of protein/DNA binding and histone modifications. MACS - Model-based Analysis of ChIP-Seq: tag distributions which represent the end of a sequenced fragment in order to resolve the centre of the bound region. MACS empirically models the amount of this shift to better determine the boundaries of the region. It works by sampling a number of high quality windows which are enriched for binding, and using these regions to estimate the shift. As an example: use Mouse ChIP: G1E CTCF binding. These are data from two different cell conditions. G1E is a mouse cell line where a specific transcription factor: we have null and restored. The transcription factor is CTCF involved in genome structure and gene regulation. Use Bowtie2 to map our data onto the mouse genome. Map this to mouse build mm9 with default parameters. Aligned in BAM format: binary alignment format. NGS Peak calling -> MACS - and then we can see how the reads align, and where they peak. We can visualize the `.BED` file of peaks. In doing ChIP-Seq analysis, there are a number of issues and biases: chromatin accessibility affects fragmentation, amplification, repetitive regions, etc. We can add a control to MACS to determine the background expectation for the number of peaks you should see, and compute a false discovery group. This is extremely important. NGS Peak calling MACS explicitly allows us to add a 'control': again - a BED file and a html report. Other tools may be more appropriate for broad histone modifications. Summary: MACs is one tool for ChIP-seq data. Controls are extremely important for accurately calling ChIP-seq peaks, but there are other tools which may be more appropriate depending on the type of data, such as broad histone modifications.

Week Four: RNA-seq and Running your own Galaxy

RNA-seq Analysis: mapping of RNA-seq data using Galaxy. Use a demonstration dataset called Human RNA-seq: CHB Encode Exercise. Grab all 5 datasets: 2 sequenced datasets (replicates) for two different cell types, and then gene annotations. Each of this is a FastQ dataset (ideally, run quality control on these datasets using FastQC). We want to look at levels of mRNA - which are isolated and fragmented using random priming to create short DNA sequences which are complimentary to the original mRNA sequences. Now we need to analyze this RNA sequencing data: which can be done in many different ways. For transcriptones - two paths we can take - we take the align then assemble: take all the reads then align them back to the reference genome. This is potentially more sensitive than the de-novo approach - which is likely to capture only highly expressed transcripts. We need to be able to align reads back to the genome in a way that is aware of splicing (with ChIP-Seq - we expected every read to align exactly back to our genome with some small differences from sequencing error). However, transcripts have been spliced: our reads may align to two regions with a large gap in-between them. To do the read-aligner, we can use Tophat: to get 4 new datasets: align summary, insertions, deletions, splice junctions and accepted hits: a mix of `.bed` and matched reads. We can run this over multiple datasets: select multiple datasets simultaneously using shift click using HG19 as the reference genome (this is a single ended dataset). Importantly, when we are using RNA-seq analysis using a reference genome, we require an aligner that is splicing aware - which means that it can handle what appear to be long deletions in the reads. Tophat is one such aligner which is available in Galaxy.

Spliced alignment provides estimates of 1.) locations of exons and 2.) splice junctions. Is this enough to know what transcripts are present? We're going to use our reference annotations as a guide: use the information from both RNA-seq and reference annotation for developing gene models. We're going to use reference annotation as a guide and run this over all 4 of our accepted hits datasets (use multiple datatsets feature in Galaxy - the BAM format ones) using Cufflinks. This will give us sets of transcripts assembled from each of our spliced map reads. Each Cufflinks job generates a couple of different datasets. We want to quantify the expression levels of each of the gene models discovered using a number called FPKN. We have a tracking name - either cuff.x or genes coming in from our reference annotation. GTF files are the actual gene models which Cufflinks has discovered. We'd like to do a statistical test of whether genes are differentially expressed between the two conditions. However, we have 4 different sets of gene annotations coming from whether the genes are differentially expressed between the two conditions. However, we have 4 different sets of gene annotations coming from the 4 different RNA-seq samples. First run a tool called cuff-merge which joins the different assemblies together. Don't forget to use the correct (chr-19) reference annotation to get a set of merged transcripts. We can now use this as our consensus gene model set for analysis - are things differentially expressed? To answer this - use Cuffdiff, which allows us to bring in all 4 of our datasets to ask whether things are differentially expressed. This gives us IDs for each of the genes and a status of whether a test was done: in some cases there is no expression data, in which case the test cannot be performed. In the cases where we have p and q value: a p-value that has been corrected for multiple testing (and then a column for whether they are expressed differently at all). Using Galaxy's filtering tools you can then extract genes which are expressed differently at different thresholds. **Summary**: Using the spliced alignments from Tophat, Cufflinks allows us to assemble transcript in Galaxy. Cufflinks will also quantify the relative abundance of each transcript in each sample. Cuffdiff performs a statistical test for differential expression based on quantitative data from multiple conditions.

Running your own Galaxy

Galaxy comes with limits on power and storage (250gb and 6 concurrent jobs) - 'Galaxy Main': the web based interface. The easiest way to get the local instance is from getgalaxy.org. Note: at the minute, it seems that Windows is not supported. You do need a version of Python 2.x (not 3.x). Ideally: clone from git:

```
git clone https://github.com/galaxyproject/galaxy
```

To get Galaxy running: `galaxy demo$ virtualenv. venv` into a virtual environment. Then: `sh run.sh` so Galaxy will automatically use this virtual environment. The server will run on localhost: 127.0.0.1:8080 - a brand new Galaxy running locally (with a limited number of tools). Use administrative access to configure access to the Galaxy instance (.ini). Create an account locally corresponding to the email address entered in the .ini, and you can then see the admin tab which allows you to administer your instance. To learn more: `galaxyproject.org/admin`. **Summary:** Galaxy is distributed through Git: `github.com/galaxyproject`. This can then be configured to run on a cluster, be more robust, etc.

Cloud computing is a model in which computing resources are available over the internet: acquired and configured quickly, and then released. There are many providers. Examples are: Amazon Web Services, Rackspace, etc. It's also possible to build local cloud computing environments (e.g. OpenStack). The recommended platform for Galaxy is `aws.amazon.com`. In particular: EC2 (elastic compute cloud - block storage and compute servers) and S3 (storage service). Galaxy itself can deal with setting it up automatically! All we really need to do is allow the cloud launch component of Galaxy to access our account. Create a new group (e.g. EC2) with certain permissions - i.e. access to EC2 and S3 (full access). We want to create a user called 'cloud-launch', and then add them to the EC2 group. This will create a new compute instance with a pre-configured version of Galaxy available by the web, through which we can then access the 'cloudman' console to configure our instance further. When authenticating: you can leave U/N blank, and use the same password as per cloudlaunch. When the cluster is ready for use, we can add additional compute nodes and nodes of different types. The grid shows the nodes of which are currently on. The instance should look the same as local or Galaxy Main, albeit now with more power and storage. To add the tools which we need: use the Galaxy Tool Shed (toolshed). This is done through the CloudMan console (Admin): there are a number of options, such as restarting services. If we add our own user here, it will modify our Galaxy configuration to allow modifications to Galaxy. If we then register a new user with the same user-name that we added an administrator, we can then access (Galaxy Main) toolshed. **Summary:** automated support provided for AWS instances which allows you to acquire as much resources as you need, and then release those resources when done.