In search of a flexible, open, and extensible visualization and query facility for biobanking.

Combining traditional and NGS sample data

Steven Githens
IUPUI
sgithens@iupui.edu

Abstract

With the explosion of low cost sequencing technologies, modern biobanks are now facing the task of interfacing their existing sample annotation data with large amounts of patient sequence data. Assembling internal and external facing user interfaces for searching, querying, and exporting of tissue data has been a fragile and tightly coupled process. In this case report, we examine the potential of using a generalized framework such as Harvest for adding data exploration functionality to the Komen Tissue Bank. An emphasis is placed on merging query capabilities across non-sequence data and NGS derived data.

General Terms biobank, sequencing, gene, data science

Keywords Harvest, Komen Tissue Bank, Python, NGS

1. Introduction

This case study involves a medium size biobank, for our examples we use the Komen Tissue Bank at the IU Simon Cancer center [1]. The Komen Tissue Bank currently has roughly 3200 physical samples, with a tragectory of doubling in size over the next decade. The central problem revolves around searching and aggregating data generated from specific samples, and how to deal with an influx of sequence data that is becoming readily available with the drop in cost of NGS technology. Banks such as this may already have information about each sample available in a relational database, but suddenly need to augment it with varying types of genomic and proteomics data being generated from sequencing specimens in the bank.

We will look at the types of data involved with this bank, their needs and use cases, and a promising technical infrastructure for harmonizing data retrieval on existing infrastructure. Additionally, we will look at future directions of this work incorporating other ways of federating and aggregating NGS queries across data stores.

2. Methods and Technologies

When looking at solutions we consider a number of of factors such as the types of data we need to aggregate and the underlying software platforms. Social and community support of development are also considered.

2.1 Categories of Data

When developing solutions to this problem space, we tend to split our data in to 2 categories: NGS data obtained from sequencing bank samples, and non-sequence data that goes with each sample. We make this distinction for the convenience of appropriating data storage and querying solutions. It will typically be the case that for most (if not all) samples, the NGS data will be much larger and complex in size, thusly requiring more thought for indexing and lookup strategies.

Figure 1 shows the general types of data that would be linked to a sample at the Komen Tissue Bank. The barcode acts as the unique identifier for each set of physical samples from an encounter. In this case an encounter consists of a woman donating blood and several cores of breast tissue. Though there are multiple cores, since they are from the same donation encounter

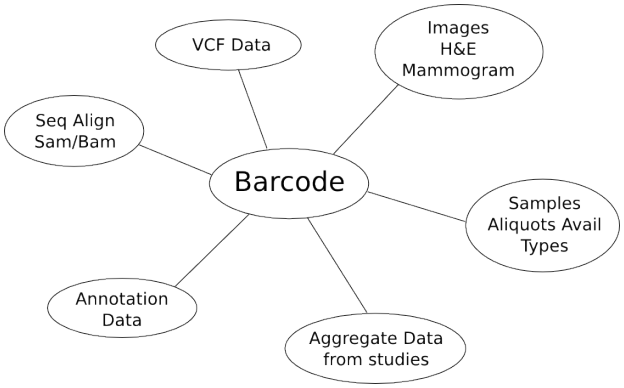

Figure 1: Sources of (meta)data for each tissue sample.

they receive the same barcode. This barcode acts as the weighted primary key for all lookups.

The non-sequence data in this example consists of the annotation data, image data, and available aliquots. The annotation data consists of a medical questionnaire with background demographics and medical history. The image data consists of mammogram DICOM images, and H&E slides that are made from extracted tissue. The available aliquots are how much sample tissue is still available from the barcode, and decreases each time tissue samples are sent out for wet lab research. The latter is an exemplary case of why we need a flexible solution that sits on top of live changing data, rather than a static viewer.

The rest is NGS data such as sequence alignment files, variant call and gene mutation data. We do acknowledge that there are many other types of data and files that are produced by NGS sequencing, but for investigating the Harvest platform libraries, we start with the VCF file format as an example.

With regard to the NGS data, there are actually several very nice web applications that allow the sort of exploration we are looking for. One example is the cBioPortal for Cancer Genomics in place at the Memorial Sloan-Kettering Cancer Center. [3]

The initial page of the cBioPortal allows the selection and filtering for various types of cancer data, as well as the input of specific genes to search for variants. After specifying the query and getting a list of patients we can see the view for a specific barcode, from which is available all the data linked to that sample, such as interesting gene mutations.

In reality, this is close to the kind of final view we want for our barcodes. The major difference is that we

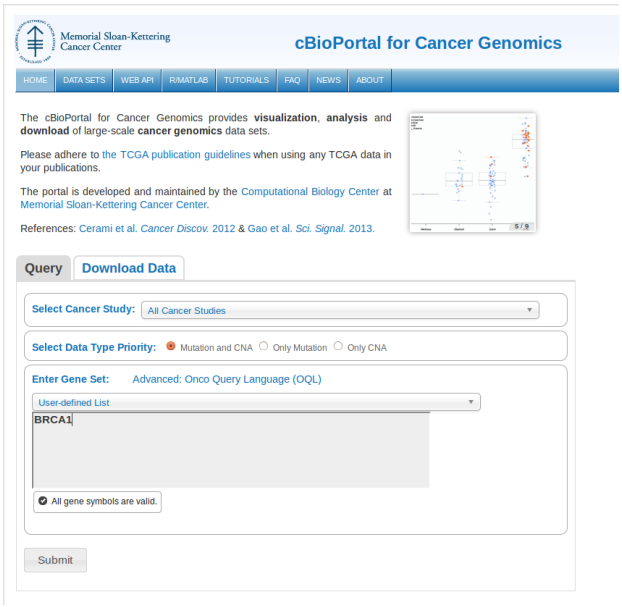

Figure 2: cBioPortal Query Home, allowing a list of genes to search and other cancer metadata.

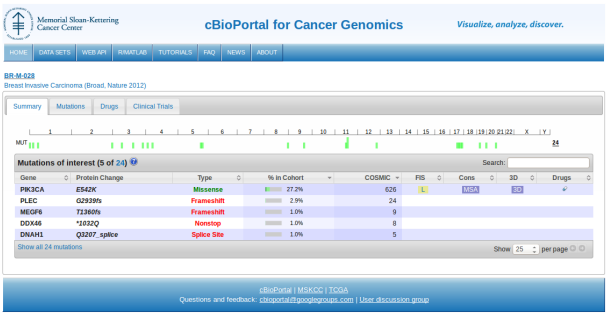

Figure 3: View of a single sample from cBioPortal highlighting mutations of interest and other data available on a tabbed user interface.

would like it housed in an infrastructure that allows easier customization and addition of metadata types to the query and view. The issue with most sites like cBioPortal is that the addition of new tabs or panes in the UI comes with the upfront cost of heavy software coding and customization. We aren't completely against doing that, but would like the entire workflow for metadata layering to be streamlined from the datastore query to the UI presentation.

2.2 Technical Considerations

To complement the data category types we like to keep in mind some other technical aspects as well. The first is whether we move from a single datasource to an ag-

gregated or federated search model, and how this impacts our search speed. For instance we start by operating on the assumption that all data is in a traditional relational database such as MySQL or Oracle, but that eventually we may need searches to run across multiple data stores.

This goes hand in hand with search speed which we like to categorize as syncronous, almost syncronous, and asyncronous. For syncronous searches, we expect them to complete in a webrowser in under 10 seconds. For "almost syncronous", a term we are making up for this paper, we consider searches that a user expects to complete quickly, but instead cause the browser to spin. Often for these the browser search can take upwards of one to two minutes to complete. Asyncronous searches are put on a queue, and the user is alerted when the search as completed.

Although we currently employ a single relational database, we keep in mind use cases for using other NoSQL datastores, as well as making calls to various web services such as those run by NCBI. For this we also note that our data accesses abundantly consist of reads rather than writes. Writes will occur when new patient data is entered, or aliquots available on a sample change, but these events occur very sparsely occured to database reads.

From a cultural perspective, it would also be nice if the software was developed by a robust open source community. In order to ensure future flexibility and extensiveness a strong group of camaraderie coupled with a modern software development platform is preferable.

3. Results

3.1 Applying the Harvest Framework

After a brief review of projects and papers, a promising candidate was found in the recently published paper on the Harvest Framework [2]. A federally funded project at the Center for Biomedical Informatics at The Children's Hospital of Philadelphia, Harvest is an open source framework built using Django and Python.

The Harvest project delivers a number of programming libraries to realize the HTML5 visualization depicted. 4 At the data layer, it builds a metadata schema on top of the Django ORM [4], that allows one to specify the necessary textual labels and relationships to generate query builders on top of relational tables. For instance, it allows you to add descriptions of what unit a particular column may be in (such as inches, meters, beats per minute, etc) or whether that information to be derived from another column. This is achieved using their Avocado library [5]. Many of these relationships can be achieved by adding configuration to their database tables, and custom widgets can be added with code. If you have a database schema that uses Django, or can easily be mapped to the Django ORM [6], then this is a straightforward process.

Along with Avocado, there is a library called ModelTree [7] that calculates all the deterministic paths in a relational schema using primary and foreign keys that allows the data exploration to be created.

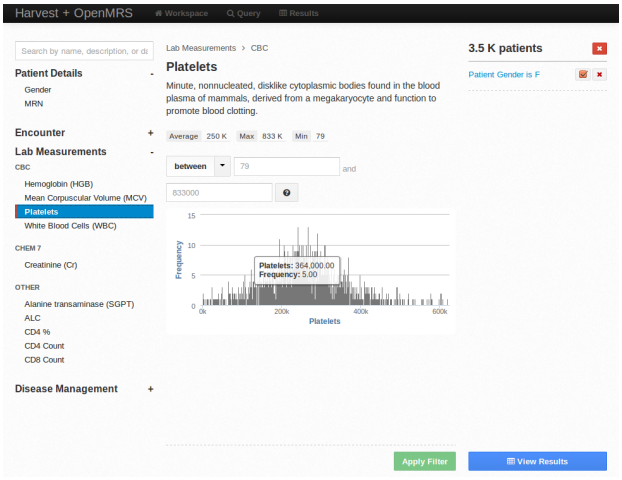

Figure 4: Open exploration of medical data using Harvest.

While this all sounds very abstract, the end result is that with just a small amount of metadata, one can create these data query UI's and exploration tools with just a small amount of metadata on top of their existing relational database schema. It is worth spending a few minutes with the online demo to see the result. [8]

3.2 VCF Format Example

Using the VCF [10] as a starting example, we can add a relational table containing columns for the fields in the VCF, plus a column for the barcode that the sequence data was derived from. It would also be useful to do some extra work on the VCF during import of the data to make queries and indexes more useful.

For example, we can add another column called variant_type to take the value of SNP, Indel, or another variant type. Our script that imports VCF data would then look at the REF and ALT fields and set the variant_type to SNP or Indel based on the two values. (ie. "G" $\rightarrow$

"C" would be SNP). With just this extra column we can begin to use the built in Harvest widgets and tooling to filter for variants with a reference and particular variant type.

The development workflow for this is as follows. After specifying a model, such as our new VCF table, Harvest contains a utility that will search your models and add their metadata to the Avocado tables using default values available from your ORM mapping. If additional tweaks or labels are needed, one can then update the metadata table rows, or programmatically create DataField and Concept objects using Python code.

4. Discussion

4.1 Future directions and data stores

If we think about the metrics from the previous section, we may become uncertain of how long this type of data storage will be scalable for a single instance of a relational database. For the VCF example, with just 3200 sequenced samples, and several million SNPs per sample we already a large amount of data. Any additional formats, and the potential desire to reference the individual's sequence alignment data would add to this.

At some point, for performance reasons, we may want to consider either sharding on several relational databases, or sharding upon multiple datastores. Some of these being web service endpoints or another type of document, key-value, or other NoSQL database.

For example, using a document oriented datastore such as MongoDB, we could easily import the VCF data as a small JSON format, and then shard Mongo instances by patient. Searching over all the nodes with a MapReduce query would could produce favorable results.

This stage of the project is currently under planning and being discussed on the Harvest discussion forums. [9]

4.2 Conclusion

Based on the criteria above, there is a strong case to prototype future tissue bank data interfaces using the Harvest framework. We believe that when used in conjunction with other local data management scripts and utilities it holds promise for harmonizing access to both NSG sequence and other annotation information associated with physical samples. Additionally, performance testing and exploration of companion relational and NoSQL datastores should be further investigated as the amount of data increases.

References

[1] Komen Tissue Bank at the IU Simon Cancer Center, http://komentissuebank.iu.edu/

[2] Pennington JW, Ruth B, Italia MJ, et al. Harvest: an open platform for developing web-based biomedical data discovery and reporting applications. J Am Med Inform Assoc doi:10.1136/amiajnl-2013-001825

[3] cBioPortal for Cancer Genomics at the Memorial Sloan-Kettering Cancer Center http://www.cbioportal.org/public-portal/

[4] Django ORM Reference, https://docs.djangoproject.com/en/1.6/topics/db/models/

[5] Avocado Metadata APIs, http://avocado.harvest.io/doc/introduction.html

[6] Django Legacy DB Documentation, https://docs.djangoproject.com/en/1.6/howto/legacy-databases/

[7] ModelTree, Dynamic QuerySet Generation without joins, http://modeltree.harvest.io/

[8] Harvest Demo using OpenMRS Medical Data, http://harvest.research.chop.edu/demo/

[9] Harvest 3.0: Document stores, continuous indexing, backends, https://groups.google.com/forum/#!topic/harveststack/bsd69roEYlg

[10] VCF Reference, http://vcftools.sourceforge.net/VCF-poster.pdf