**Revised SVFX Attempt Pipeline**

Note: All pertinent files have been placed in /ysm-gpfs/pi/gerstein/jv485/final\_pipeline/ This will be treated as the base directory for file location references.

Necessary python package installations:

pip install scikit-allel –-user

pip install pyBigWig –-user

pip install argparse –-user

pip install sklearn --user

Necessary module loads are in the appropriate job bash files.

**Overview:**

1. Input SV’s are in VCF form from the 1KG database and BED form from the cancer database
2. BRCA and 1KG input files were then parsed and sorted into 3 “.coord” files (these are simply tab-separated value files containing the chromosome, start, and end coordinates of each SV below 200,000 BP in length). The BRCA coordinate files are located in coord\_files/cancer\_[del, dup, inv].txt while the 1KG coord files are located in 1kg\_files/1kg\_[del, dup, inv].txt
3. Features were then extracted from these coordinate files and various genome-data files through generate\_feature\_matrix.py. This step is further detailed later in the document.
4. Input feature matrices were then passed through a scikit-learn random forest implementation in rf\_model.py. This step is further detailed later in the document.
5. Ancillary data analysis metrics (e.g. AUROC/PRC, feature distributions, clustering) were provided by a number of other small scripts, which will be listed later in the document.

**Feature Matrix Generation:**

Broadly, generate\_feature\_matrix.py (or gfm for short) takes a list of SV coordinates, as well as a number of feature files, as inputs and returns a feature matrix as an output. The code is fairly well-commented, so I will instead go over the program’s intended usage structure, starting at a high level and getting more specific.

I sort feature files into two types: bigwig files (.bw) and “overlap files.” The first type is self explanatory – the second type is the collection of files that contain coordinates for various features (e.g. CTCF areas, cancer gene coordinates).

Multiple feature-calculation methods were used in different iterations of the program – I will go over the main two:

In the first iteration, there were 4 features calculated per SV per bigwig file, and 2 features calculated per SV per overlap file. For each bigwig file, I calculated the mean value across the SV window, the mean absolute deviation (MAD) across the SV window, the mean value across the “extended” SV window (the SV window with an additional 2000 nucleotides upstream and downstream), and the MAD across the extended SV window.

For each overlap file, I simply calculated the coverage ( for both the normal and extended SV windows.

In the second iteration, I made two changes: I removed the extended window calculations (they were too strongly correlated with the normal windows’ calculations) and added a “maximum” feature. The maximum for bigwig files was the highest mean over disjoint 10-BP sub-windows, while the maximum for overlap was the highest coverage over disjoint 10-BP windows.

The features I used can be found in feature\_files\_README, while the specific files I used can be found in my sample job scripts. Note, though, that this program is generalized, so any feature file of accepted formats (bigwig/BED) can be added to the job script and used to construct the matrix.

A sample gfm job script can be found in generate\_feature\_matrix.sh. For the purposes of this project, the usage is:

python generate\_feature\_matrix.py [-f if this is cancer data] --svtype [DEL, DUP, or INV] -c [input SV coordinate file] -o [output matrix filename] –b [bigWig files] –g [overlap files]

The additional flags described in the program’s main method are intended for various general usage cases, and can be ignored.

Note: gfm must be used separately on cancer and 1kg data, but cancer and 1kg matrices must be combined for the random forest training input.

Splitting Gfm:

Gfm will take an unreasonable amount of time as both the size and number of SV’s increases. To sidestep this, use split\_gfm.py. This script simply splits a given coordinate file into multiple temporary files with 100 SV’s each, copies a template job into individualized jobs for each of them, compiles all these jobs into a job list, and batches them using Dead Simple Queue. When the Dead Simple Queue is finished, the resulting matrices can simply be recombined by the keyword passed to split\_gfm.py. (Note: using this script will not preserve the order from the coordinate file)

The usage is as follows:

python split\_gfm.py [input file] [keyword] [SV type] [job template]

For example, suppose we had an input file 1kg\_del.py, and a job template (e.g. .sh file for gfm, but without specific input/output/svtypes mentioned) called template\_job.sh shown below (Note: the template job MUST be formatted as below):

[SBATCH options and module loading]

…

python generate\_feature\_matrix.py --svtype TYPE\_SV -c INPUT\_FILE -o OUTPUT\_FILE -b [some bigWig files] –g [some overlap files]

Then, [input file] would be 1kg\_del.py, [SV type] would be DEL, and [job template] would be template\_job.sh.

[keyword] is simply the string that will be the root for all of the subfiles created after splitting – it’s used to enable easy processing of all of the split files and their jobs/matrices. The only guideline for the keyword is that the string SHOULD NOT appear as a substring for any other files in the directory, or else the scripts will group those files with the split files for processing.

**Generating Randomized SV’s**

One issue we noticed with training on 1KG vs. diseased SV’s is that if features have any length correlation, then differing length distributions between classes will bias the model. To avoid this, we replaced 1KG SV’s, which had a much lower length distribution, with a set of SV’s of exactly the same length distribution as the cancer SV’s, randomly scattered throughout the genome.

To generate randomized SV’s, you first need to supply an input file containing the lengths of the SV’s to be generated, one per line (this is quite easy to get from a coordinate file – simply use awk ‘{print $3 - $2}‘ > [length file]).

You’ll also need a fasta index file for the reference genome (use samtools faidx [fasta] for this). The usage is:

python SVcall.py [length file] [index file] [output filename]

The output is a coord file of randomized SV coordinates (tab-separated chromosome, start, and end).

**Training the Random Forest**

To train a random forest classifier on a feature matrix and get an output file with scores for each set of coordinates, use rf\_model.py.

The input matrix must have the cancer and the 1KG SV data, and it must have all cancer SV’s before all 1KG SV’s.

Usage is as follows:

#!/bin/bash

#SBATCH --partition=general

#SBATCH --ntasks=1 --nodes=1

#SBATCH --mem-per-cpu=10000

#SBATCH --cpus-per-task=6

#SBATCH --time=100:00:00

#SBATCH --mail-user=jagath@caltech.edu

#SBATCH --mail-type=ALL

module load Python

python rf\_model.py -i [input feature matrix] -d [file with indices to ignore for training, one per line] -t [index with target values] -n [number of trees to use in forest] -c [number of cancer SV’s] -l [number of total SV’s] -o [base name for output files (no extension)]

This will output the predictions (in the output name + \_predictions.txt) as well as a number of serialized object files.

**Instructions for going from provided duplication files to SV scores:**

1. sbatch arif\_cancer\_generate\_feature\_matrix.sh
2. sbatch arif\_1kg\_generate\_feature\_matrix.sh
3. vim 1kg\_dup\_matrix.tsv
4. dd (delete first line)
5. cp cancer\_dup\_matrix.tsv dup\_feature\_matrix.tsv
6. cat 1kg\_dup\_matrix.tsv >> dup\_feature\_matrix.tsv

At this point, you will have generated the feature matrix. (Note: the 4 histone marks will not have their features averaged across cell lines here – I did that manually very quickly in Excel; I didn’t add that feature to generate\_feature\_matrix to keep the usage general and avoid excessive instructions. Also, the matrix will include GC content and Allele Frequency no matter what; this doesn’t matter, as you can easily choose which features to include in training in the next step)

1. printf “0\n1\n2\n3\n4\n5” > non\_training\_indices.txt
2. sbatch rf\_model.sh

Once this job has completed, you should see dup\_predictions.txt and dup\_predictions.pkl, which contain the predictions (the text file contains 5 columns: the prediction, the label, the chromosome, the start coordinate, and the end coordinate. The pkl file contains the serialized dictionary of predictions (which can be loaded into a python program using joblib.load(filename). The dictionary, at key i, contains the prediction for the ith SV in the matrix).

The 10 models are also stored in a pkl file (dup\_ten\_models.pkl), which can similarly be loaded into a Python program for use in scoring further data.

**Scoring SV’s with existing models and interpreting results**

I use joblib’s object serialization to store models in .pkl format. More specifically, the output file from the rf code produces a .pkl file that contains a list of the 10 trained models.

To score a set of SV’s using a serialized batch of trained models, first generate the feature matrix for the SV’s (using the methods I described at the beginning of this document). Be sure that the matrix is *formatted in the same way as the matrices used to train the serialized models*, or else the scoring program will either reject the matrix or return nonsensical results.