SAVI Manual

Oliver (oe2118@cumc.columbia.edu)
August 2015

Introduction

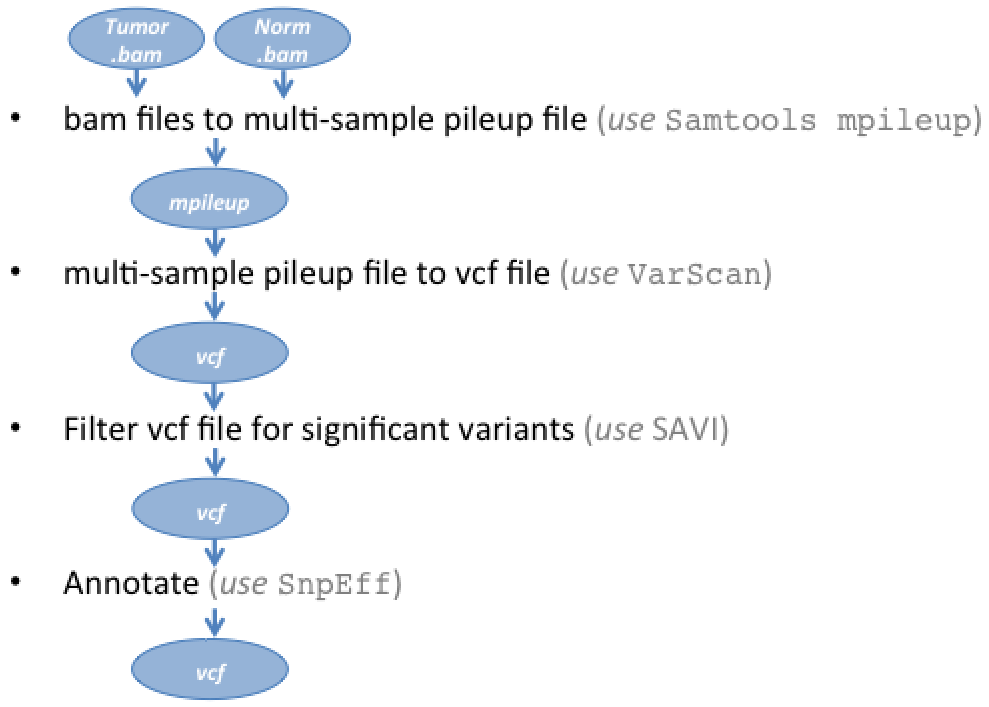
SAVI, *statistical algorithm for variant identification*, is a program for calling variants in genomic sequencing data. Why run SAVI? In a word, SAVI is for finding needles in haystacks. It can boil a very large dataset into a small list of mutations. In bioinformatics, *calling variants* can mean two different things: (1) simply enumerating all differences between some sequence data and a reference; and (2) determining which of those differences are significant and not likely to be error. SAVI does the later, while a program like `samtools mpileup` will do the former. In practice, SAVI is a way of sifting through large amounts of data to pull out the significant mutations using a Bayesian probability model. A common use case is identifying deleterious mutations in cancer, given normal and tumor sequence data—an often-encountered problem in bioinformatics. The output of SAVI is a list of candidate genomic alterations each annotated with a probability of how likely it is to be real.

SAVI works with standard bioinformatic file formats. As input, the SAVI pipeline takes bam files and it produces vcf, as well as tsv, files as output. In the output vcf files, the SAVI probabilities are added in the INFO field.

If you're interested in the mathematical underpinings of SAVI, you can read about it in this BMC Systems Biology paper [1].

General Variant Calling Pipelines in Bioinformatics

Let's look at a schematic for a general variant calling pipeline in bioinformatics, paying attention to the input/output file formats as data moves through it:


Tumor .bam
Norm .bam
bam files to multi-sample pileup file (use Samtools mpileup)
mpileup
multi-sample pileup file to vcf file (use VarScan)
vcf
Filter vcf file for significant variants (use SAVI)
vcf
Annotate (use SnpEff)
vcf

To flesh this out, a typical workflow might be:

- align reads to reference with bwa or bowtie to produce bam files
- produce a (multi-sample) pileup file with `samtools mpileup`
- change the pileup file into vcf format with bcftools or VarScan
- filter the vcf for significant variants
- annotate the filtered vcf file with SnpEff.

How the filtering is done is the crux of this problem and determines whether or not this pipeline will be lousy or good.

The SAVI Pipeline

The SAVI pipeline is organized into 5 main steps and loosely follows the schema of the general variant calling pipeline described in the previous subsection. Confusingly, we use *"SAVI"* to mean two things: (1) as a shorthand for the pipeline as a whole; and (2) more precisely, to refer to the all-important step 4 of this multi-step pipeline—the one that filters significant variants from insignificant ones. The steps are:

1. samtools mpileup: bams to mpileup (input: bam files; output: pileup file)
2. pileup2multiallele_vcf: mpileup to vcf (input: pileup file; output: vcf file)
3. Savi: Make Prior (input: vcf file; output: prior files)

4. Savi: Run Savi (input: vcf file + prior files; output: vcf files)
5. SnpEff Annotation (input: vcf files; output: vcf and tsv files)

In plain language:

- **Step 1** produces a pileup file from multiple bam files—in the simplest case, paired normal-tumor samples.
- **Step 2** converts the pileup file into vcf format.
- **Step 3** generates a prior from the data, but by default it is skipped to save time and a standard diploid prior is used.
- **Step 4** is the SAVI proper part, which takes a large vcf with many variants and outputs a smaller vcf containing the variants it believes are present annotated with a probabilities.
- **Step 5** annotates these variants, providing information about what gene features the variant intersects (is it in an intron, exon, etc.?); whether it's in the Cosmic database; and so on.

Dependencies

Switching from theory to practice, to run SAVI the following programs must be in your `PATH`:

- python
- java
- Samtools v1.2
- SnpEff v4.1 C (i.e., `which snpEff.jar` must return a path)
- tabix
- bgzip
- vcflib

The following Python packages are required:

- scipy

Notes

To get vcflib, clone the repository using the `--recursive` flag because it contains sub-repositories:

```
git clone --recursive https://github.com/ekg/vcflib.git
```

Finally, you must put the vcflib binaries in your `PATH` for SAVI to function. Did you get it right? Test it by typing:

```
vcffilter
```

It should return a help screen.

Also note that you'll have to set up a reference for SnpEff. SAVI assumes this reference is called *hg19* by default, although you can override this with a flag.

Installation

Once you've taken care of the dependencies, installing SAVI is simple. First, clone the repository:

```
git clone https://github.com/RabadanLab/SAVI.git
```

Then `make` the binaries in the `SAVI/bin` directory:

```
cd SAVI/bin
make
```

Additional Files

SAVI needs a reference fasta file (whatever you mapped your bams to) and its faidx index. It is also helpful to use various vcf files to provide additional annotation. You can download all of this on the Rabadan Lab homepage at:

rabadan.c2b2.columbia.edu/public/savi_resources/

You'll find the following files—a human reference, hg19:

- hg19_chr.fold.25.fa
- hg19_chr.fold.25.fa.fai

And various annotating vcfs:

- dbSnp138.vcf - dbSnp 138
- cbio.fix.sort.vcf - cBio variants
- CosmicCodingMuts.v72.May52015.jw.vcf - Cosmic variants
- CosmicNonCodingVariants.v72.May52015vcf - Cosmic variants
- 219normals.cosmic.hitless100.noExactMut.mutless5000.all_samples.vcf - Rabadan Lab supernormal
- meganormal186TCGA.fix.sort.vcf - Rabadan Lab TCGA supernormal

Variant Calling Protocol

SAVI is *not* sufficient to get a reliable list of candidate mutations.

One source of false positive variants which could end up in your final report which SAVI does not handle is clonal reads—a systematic of some deep sequencing experiments. If they are in your bam files, variants at these positions will have artifically inflated depths and erroneously appear to be significant. The recommended course of action is to correct this systematic with Picard (broadinstitute.github.io/picard/). You have to take care of this before you begin or else you'll end up with noise in your final signal. The same caveat applies to adapter contamination, which the program Cutadapt (cutadapt.readthedocs.org/en/stable/) can lessen.

Before you run SAVI, you should follow these steps:

- first run fastqc (www.bioinformatics.babraham.ac.uk/projects/fastqc/) to check the quality of your fastq files
- remove low quality sequences and remove adapter contamination with cutadapt (cutadapt.readthedocs.org/en/stable/), then re-check quality
- map your reads with bwa (bio-bwa.sourceforge.net)
- run picard (broadinstitute.github.io/picard/) to remove PCR duplicates

Usage Examples

The most common use case for SAVI is paired normal-tumor bam files (mapped to the same reference, of course) where the goal is to find the significant mutations in the tumor sample. Before we run SAVI, we need to make sure of the following:

- the reference fasta we're providing to SAVI is the same one to which we've mapped our bam files
- we've indexed the reference fasta with `samtools faidx`
- we've sorted our bams files via `samtools sort`
- we've indexed our sorted bam files via `samtools index`

Got it? Good! Here's an example command running SAVI for chromosome 1:

```
SAVI/savi.py --bams normal.bam,tumor.bam --names NORMAL,TUMOR \
--ref hg19_chr.fold.25.fa --region chr1 --outputdir outputdir/chr1 \
--annvcf dbSnp138.vcf,meganormal186TCGA.fix.sort.vcf,...
```

By convention, if you have a normal sample, put it first in the list as in: *normal.bam,tumor.bam*. By default, the program compares all samples in the list to the first one.

In practice, we'd want to run SAVI for every chromosome in a loop:

```
for i in chr{1..22} chr{X,Y,M}; do
    # command here
done
```

Another use case is a tumor-only bam file. Here's a sample command again for chromosome 1:

```
SAVI/savi.py --bams tumor.bam --ref hg19_chr.fold.25.fa \
--region chr1 --outputdir outputdir/chr1 \
--annvcf dbSnp138.vcf,meganormal186TCGA.fix.sort.vcf,...
```

In this case, SAVI won't be able to call somatic variants. Instead, it will merely tell us what variants it thinks are present—a much longer list than if we had the normal sample for comparison.

Common Problems

- Are your bams sorted by position? `samtools mpileup` requires "position sorted alignment files."
- Did you index your bams to produce bai files? Samtools won't be able to extract regions of your bam files if they have not been indexed.
- Are both the tumor and normal bam files mapped to the same reference?
- Are you using the proper region identifiers? Some references use *chr1* while others merely use *1*.

Parameter Tuning

It's a tautology to state that the output of the pipeline is highly dependent on which parameters are fed into the many programs comprising it. Yet it's important to emphasize this all the same and, for this reason, you may find yourself playing with input flags or even going into the wrapper to change some parameters as you see fit. As a simple example, the default depth cutoff is 10. However, you might want to reduce this if your sample has low depth or turn it up if your sample has high depth. There is a trade-off between automation and flexibility and no robot can choose your parameters for you, although the wrapper uses sensible defaults.

An In-Depth Look at the Pipeline Wrapper

Step 1

Step 1 is a wrapper for `samtools mpileup`. It converts your bam files to pileup format, and it post-processes the pileup file to give us only lines where there are variant-containing reads (unless you're building a prior, as discussed in Step 3). It also does some basic read-depth filtering, throwing away low coverage depth positions, and quality filtering via `samtools mpileup`'s baseQ and mapQ cutoffs. You can change these ad-hoc filters if you like, but we use them because it speeds up the runtime significantly and reduces noise. Note that, with `samtools mpileup`'s approach, all samples are merged from the get-go.

Step 2

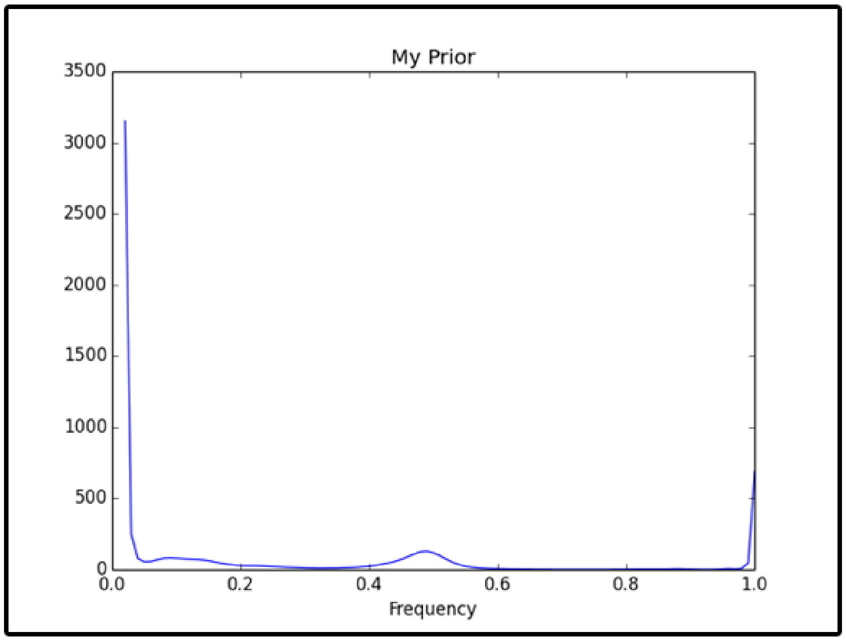
Step 2 converts pileup format to vcf format with minimal to no filtering. Every merged sample has at least the following sub-fields in the genotype fields:

- **RBQ** Average quality of reference-supporting reads
- **ABQ** Average quality of variant-supporting reads
- **RDF** Depth of reference-supporting reads on forward strand
- **RDR** Depth of reference-supporting reads on reverse strand
- **ADF** Depth of variant-supporting reads on forward strand
- **ADR** Depth of variant-supporting reads on reverse strand

This is important because the raw input to compute the SAVI statistics is the variant depth counts.

Step 3

Step 3 wraps a script that makes the Bayesian priors. By default, this step is skipped to save time and a standard diploid prior is used instead of generating one from the data. If you run this step, the input is a vcf file with both variant and non-variant positions alike. The prior building program is iterative and the default settings run 10 iterations for each sample, starting with the vcf file and a uniform prior. The result should be priors that look roughly like this:



with a bump at a frequency of 50% produced by heterozygous variants in the diploid human genome.

Step 4

Step 4 wraps the script which computes the SAVI statistics. In the simple case of comparing sample 2 (tumor) to sample 1 (normal), SAVI adds the following sub-fields to the vcf INFO field:

- **S1_P** Savi presence call boolean for sample 1
- **S1_PF** Savi posterior for presence or absence for sample 1
- **S2_P** Savi presence call boolean for sample 2
- **S2_PF** Savi posterior for presence or absence for sample 2
- **P1_F** Savi frequency for sample 1
- **P1_L** Savi frequency lower bound for sample 1
- **P1_U** Savi frequency upper bound for sample 1
- **P2_F** Savi frequency for sample 2
- **P2_L** Savi frequency lower bound for sample 2
- **P2_U** Savi frequency upper bound for sample 2
- **PD21_F** Savi frequency delta for sample 2 vs 1
- **PD21_L** Savi frequency delta lower bound for sample 2 vs 1
- **PD21_U** Savi frequency delta upper bound for sample 2 vs 1

The program does two things. The first is to filter for variants that are present in either sample. The field `S1_P`, for instance, is a boolean that gives a presence call for sample 1, while `S1_PF` gives the posterior for either the presence or absence. The default filter in our two sample example looks like this:

```
( S1_P = 1 & S1_PF < 1e-6 ) | ( S2_P = 1 & S2_PF < 1e-6 )
```

The second thing the program does is to add the rest of the stats without further filtering. Later, in Step 5, we apply an additional filter on `PD21_L`, the lower bound of the frequency delta, to find significant variants in one of the reports.

If we had multiple samples, we could make any comparisons we like by adding this information in the `--compsamp` flag. For example,

```
--compsamp 2:1,3:1,3:2
```

would compare 2 to 1, 3 to 1, and 3 to 2.

Step 5 and Pipeline Output

Step 5 is a wrapper for SnpEff along with some commands to filter the final reports in various ways. After annotating genomic features with SnpEff, Step 5 loops over the comma-delimited list of vcf files given in the `--annvcf` flag, so you can provide as many external annotations as you like. We typically use a series of vcfs like:

```
dbSnp138.vcf, \
CosmicVariants_v66_20130725.vcf, \
219normals.cosmic.hitless100.noExactMut.mutless5000.all_samples.vcf, \
cbio.fix.sort.vcf, \
meganormal186TCGA.fix.sort.vcf
```

to annotate variants in dbSnp, Cosmic mutations, and variants in our own “super normal”—a list of non-significant mutations we’ve compiled which can appear as significant due to systematic errors. This is an important negative control.

Step 5 also produces reports, in both vcf and tsv format, with various filters:

- *report.all* - the product of the filters heretofore applied
- *report.coding* - filter for variants in the coding region
- *report.coding.somatic* - filter for variants in the coding region; discard variants in the meganormals; discard common dbSnps; discard variants where the normal sample has variant-supporting reads above a certain depth threshhold

- *report.coding.PDfilter* - filter for variants in the coding region; filter such that the frequency delta lower bound for tumor vs normal is greater than zero; other filters

In practice, you'll probably want to open up Excel and do some of your own filtering on top of the default filters.

savi.py Options

```
Usage: savi.py [-h] [--bams BAMS] [--ref REF] [--outputdir OUTPUTDIR]
               [--region REGION] [--names NAMES] [--compsamp COMPSAMP]
               [--steps STEPS] [--ann ANN] [--memory MEMORY]
               [--scripts SCRIPTS] [--mindepth MINDEPTH] [--minad MINAD]
               [--mapqual MAPQUAL] [--maxdepth MAXDEPTH] [--s1adpp S1ADPP]
               [--minallelefreq MINALLELEFREQ] [--annvcf ANNVCF]
               [--buildprior BUILDPRIOR] [--prior PRIOR]
               [--prioriterations PRIORITERATIONS] [--presence PRESENCE]
               [--conf CONF] [--precision PRECISION] [--noindeldepth]
               [--rdplusad] [--index INDEX] [--noncoding] [--noclean]
               [--noerror] [--verbose] [--superverbose]

Arguments:

  -h, --help            show this help message and exit
  --bams BAMS, -b BAMS  comma-delimited list of bam files (by convention list
                        the normal sample first, as in: normal.bam,tumor.bam)
                        (.bai indices should be present)
  --ref REF             reference fasta file with faidx index in the same
                        directory
  --outputdir OUTPUTDIR, -o OUTPUTDIR
                        the output directory (default: cwd)
  --region REGION, -r REGION
                        the genomic region to run SAVI on (default: full
                        range) (example: chr1 or chr1:1-50000000)
  --names NAMES         sample names in a comma-delimited list, in the
                        corresponding order of your bam files (default: names
                        are numerical indicies)
  --compsamp COMPSAMP, -c COMPSAMP
                        comma-delimited list of colon-delimited indices of
                        samples to compare with savi (default: everything
                        compared to sample 1) (example: 2:1 would compare the
                        second bam file to the first) (example: 2:1,3:1,3:2
                        would compare the second to the first, the third to
                        the first, and the third to the second)
  --steps STEPS         steps to run (default: 1,2,4,5 (i.e., all except prior
                        generation))
```

```
--ann ANN             name of the SnpEff genome with which to annotate
                      (default: hg19)
--memory MEMORY       the memory for the (SnpEff) Java virtual machine in
                      gigabytes (default: 6)
--scripts SCRIPTS     location of scripts dir (directory where this script
                      resides - use this option only if qsub-ing with the
                      Oracle Grid Engine)
--mindepth MINDEPTH   the min tot read depth required in at least one sample
                      - positions without this wont appear in pileup file
                      (default: 10). Where the filtering occurs: samtools
                      mpileup post-processing
--minad MINAD         the min alt depth (AD) in at least one sample to
                      output variant (default: 2). Where the filtering
                      occurs: samtools mpileup post-processing
--mapqual MAPQUAL     skip alignments with mapQ less than this (default:
                      10). Where the filtering occurs: samtools mpileup
--maxdepth MAXDEPTH   max per-BAM depth option for samtools mpileup
                      (default: 100000)
--s1adpp S1ADPP       for filtered report, require the sample 1 (normal) alt
                      depth per position to be less than this (default: 3)
                      (note: this is NOT sample1 alt depth of the given alt
                      but, rather, at the given position). Where the
                      filtering occurs: generating report.coding.somatic
--minallelefreq MINALLELEFREQ
                      Sgt1MAXFREQ (the allele frequency of any sample not
                      including the first one, assumed to be normal) is
                      greater than this (default: 4) Where the filtering
                      occurs: generating the PD.report file.
--annvcf ANNVCF       comma-delimited list of vcfs with which to provide
                      additional annotation (default: none). Where it's
                      used: SnpSift
--buildprior BUILDPRIOR
                      starting input prior when building the prior if step 3
                      is invoked (default: bin/prior_unif01)
--prior PRIOR         prior to use if step 3 is not run (default:
                      bin/prior_diploid01)
--prioriterations PRIORITERATIONS
                      the number of iterations for the prior build, if step
                      3 is run (default: 10)
--presence PRESENCE   the SAVI presence posterior (default: 1e-6). Where
                      it's used: step 4
--conf CONF           the SAVI conf (default: 1e-5). Where it's used: step 4
--precision PRECISION
                      the SAVI precision (default: 0). Where it's used: step
                      4
--noindeldepth        do not include include indel reads in total depth
                      count (SDP) (default: off) (note: asteriks in mpileup
                      are included irrespective of this flag). Where it's
                      used: step 2
--rdplusad            use reference-agreeing reads plus alternate-calling
                      reads (RD+AD) rather than total depth (SDP) as input
                      to savi (default: off). Where it's used: step 2
--index INDEX         an index used in naming of output files (default: 0)
--noncoding           use snpEff to find all transcripts, not just only
                      protein transcripts (default: off). Where it's used:
                      step 5
--noclean             do not delete temporary intermediate files (default:
                      off)
--noerror             do not check for errors (default: off)
--verbose, -v         echo commands (default: off)
--superverbose        print output of the programs called in each step
                      (default: off)
```

References

[1] Vladimir Trifonov, Laura Pasqualucci, Enrico Tiacci, Brunangelo Falini and Raul Rabadan. *SAVI: a statistical algorithm for variant frequency identification. BMC Systems Biology* 2013, 7(Suppl 2):S2 doi:10.1186/1752-0509-7-S2-S2.