SomaticSeq Documentation

Li Tai Fang / li_tai.fang@roche.com

August 27, 2018

Contents

1 Introduction

SomaticSeq is a flexible post-somatic-mutation-calling algorithm for improved accuracy. It is compatible with 10+ somatic mutation caller(s). Any combination of them can be used to obtain a combined call set with sequencing features extracted into TSV and VCF files. In addition, SomaticSeq uses machine learning (adaptive boosting) to distinguish true mutations from false positives from that call set. The mutation callers we have incorporated are MuTect/Indelocator/MuTect2, VarScan2, JointSNVMix, SomaticSniper, VarDict, MuSE, LoFreq, Scalpel, Strelka, and TNscope. You may incorporate some or all of those callers into your own pipeline with SomaticSeq.
The manuscript, An ensemble approach to accurately detect somatic mutations using SomaticSeq, was published in Genome Biology 2015, 16:197. The SomaticSeq project is located at *https://github.com/bioinform/somaticseq*. There have been some major improvements in SomaticSeq since that Genome Biology publication in 2015.
The wrapper script somaticseq/run_somaticseq.py and its parallelized cousin somaticseq_parallel.py can 1) train the call set into a classifier, 2) predict high-confidence somatic mutations from the call set based on a pre-defined classifier, or 3) default to consensus mode, i.e., extract sequencing features and output the TSV and VCF files, and then label the calls (i.e., PASS, LowQual, or REJECT) based on majority vote of the tools.

1.1 Dependencies

- Python 3, plus pysam (v0.14.1), numpy (v1.14.3), and scipy (v1.1.0). The versions in parentheses are in our docker images and validated to work, though other versions should work, too.
- R, plus the ada package in R.
- BEDTools (if inclusion and/or an exclusion region files are supplied, and/or running somaticseq_parallel.py instead of somaticseq/run_somaticseq.py)
- Optional: dbSNP in VCF format (if you want to use dbSNP membership as a part of the training).
- At least one of MuTect/Indelocator/MuTect2, VarScan2, JointSNVMix2, SomaticSniper, VarDict, MuSE, LoFreq, Scalpel, Strelka2 and/or TNscope. Those are the tools we have incorporated in SomaticSeq. If there are other somatic tools that may be good addition to our list, please make the suggestion to us.

1.2 Docker images

SomaticSeq and most somatic mutation callers we have incorporated are dockerized.

- SomaticSeq: https://hub.docker.com/r/lethalfang/somaticseq
- MuTect2: https://hub.docker.com/r/broadinstitute/gatk
- VarScan2: https://hub.docker.com/r/djordjeklisic/sbg-varscan2
- JointSNVMix2: https://hub.docker.com/r/lethalfang/jointsnvmix2
- SomaticSniper: https://hub.docker.com/r/lethalfang/somaticsniper
- VarDict: https://hub.docker.com/r/lethalfang/vardictjava
- MuSE: https://hub.docker.com/r/marghoob/muse
- LoFreq: https://hub.docker.com/r/marghoob/lofreq
- Scalpel: https://hub.docker.com/r/lethalfang/scalpel
- Strelka2: https://hub.docker.com/r/lethalfang/strelka

2 How to run SomaticSeq

The *somaticseq/run_somaticseq.py* calls a series of programs and procedures **after** you have run your individual somatic mutation callers, and *somaticseq_parallel.py* is a wrapper script that allows parallel processing. Section 5 will teach you how to run those mutation callers that we have been dockerized. It also includes ways to create semi-simulated training data that can be used to create SomaticSeq classifiers. In the next section, we will describe the workflow in this wrapper script in detail.
Both paired and single modes are supported, although single mode is not as well validated scientifically as the paired mode. To see the required and optional input files and parameters to *somaticseq_parallel.py*:

```
# See the global input parameters
somaticseq_parallel.py --help

# Input parameters for paired-sample mode (i.e., tumor-normal)
somaticseq_parallel.py paired --help

# Input parameters for single-sample mode
somaticseq_parallel.py single --help
```

2.1 SomaticSeq Training Mode

To create SomaticSeq classifiers, you need a VCF file containing true SNVs and a VCF file containing true INDELs. There is also an option to include a list of regions to include and/or exclude from this exercise. The exclusion or inclusion regions can be VCF or BED files. An inclusion region may be subset of the call sets where you have validated their true/false mutation status, so that only those regions will be used for training. An exclusion region can be regions where the "truth" is ambigious. All the variants in the truth VCF files are assumed to be true positives. Every mutation call not in the truth VCF files is assumed to be false positives (as long as the genomic coordiante is in inclusion region and not in exclusion region if those regions are provided).
All the output VCF files from individual callers are optional. Those VCF files can be bgzipped if they have .vcf.gz extensions. It is imperative that you will use the same parameter for prediction as you do for training.

```
# An example command for SomaticSeq Training.
somaticseq_parallel.py \
--somaticseq-train \
--output-directory  $OUTPUT_DIR \
--genome-reference  GRCh38.fa \
--truth-snv         truePositives.snv.vcf \
--truth-indel       truePositives.indel.vcf \
--inclusion-region  genome.bed \
--exclusion-region  blacklist.bed \
--threads           12 \
paired \
--tumor-bam-file    tumor.bam \
--normal-bam-file   matched_normal.bam \
--mutect2-vcf       MuTect2/variants.vcf \
--varscan-snv       VarScan2/variants.snp.vcf \
--varscan-indel     VarScan2/variants.indel.vcf \
--jsm-vcf           JointSNVMix2/variants.snp.vcf \
--somaticsniper-vcf SomaticSniper/variants.snp.vcf \
--vardict-vcf       VarDict/variants.vcf \
--muse-vcf          MuSE/variants.snp.vcf \
--lofreq-snv        LoFreq/variants.snp.vcf \
--lofreq-indel      LoFreq/variants.indel.vcf \
--scalpel-vcf       Scalpel/variants.indel.vcf \
--strelka-snv       Strelka/variants.snv.vcf \
--strelka-indel     Strelka/variants.indel.vcf
```

For the command's argument placement, caller output and bam files are input "after" *paired* or *single* option. Everything else goes before, e.g., reference, ground truths, resources such as dbSNP and COSMIC, etc.
Parallel processing is achieved by splitting the inclusion BED file into a number of sub-BED files of equal region sizes, named 1.th.input.bed, 2.th.input.bed, ..., n.th.input.bed. Then each process will be run using each sub-BED file as the inclusion BED file. If there is no inclusion BED file in the command argument, it will split the reference.fa.fai file instead.
SomaticSeq supports any combination of the somatic mutation callers we have incorporated into the workflow. SomaticSeq will run based on the output VCFs you have provided. It will train for SNV and/or INDEL if you provide the truePositives.snv.vcf and/or truePositives.indel.vcf file(s) and invoke the *--somaticseq-train* option. Otherwise, it will fall back to the simple caller consensus mode.

2.2 SomaticSeq Prediction Mode

Make sure the classifiers (.RData files) are supplied, Without either of them, or it will fall back to the simple caller consensus mode.

```
# The *.RData files are trained classifier from the training mode.
somaticseq_parallel.py \
--classifier-snv     Ensemble.sSNV.tsv.ntChange.Classifier.RData \
--classifier-indel   Ensemble.sINDEL.tsv.ntChange.Classifier.RData \
--output-directory   $OUTPUT_DIR \
--genome-reference   GRCh38.fa \
--inclusion-region   genome.bed \
--exclusion-region   blacklist.bed \
--threads            12 \
paired \
--tumor-bam-file     tumor.bam \
--normal-bam-file    matched_normal.bam \
--mutect2-vcf        MuTect2/variants.vcf \
--varscan-snv        VarScan2/variants.snp.vcf \
--varscan-indel      VarScan2/variants.indel.vcf \
--jsm-vcf            JointSNVMix2/variants.snp.vcf \
--somaticsniper-vcf  SomaticSniper/variants.snp.vcf \
--vardict-vcf        VarDict/variants.vcf \
--muse-vcf           MuSE/variants.snp.vcf \
--lofreq-snv         LoFreq/variants.snp.vcf \
--lofreq-indel       LoFreq/variants.indel.vcf \
--scalpel-vcf        Scalpel/variants.indel.vcf \
--strelka-snv        Strelka/variants.snv.vcf \
--strelka-indel      Strelka/variants.indel.vcf
```

2.3 Consensus Mode

Same as the commands previously, but without including classifiers or invoking *–somaticseq-train*. Without those information, SomaticSeq will forgo machine learning, and fall back into a simple majority vote.

3 Use SomaticSeq as a Python library

Section 2 described how to use SomaticSeq as a standalone software, but SomaticSeq can also be treated as a python library for your own software.

3.1 somaticseq.somaticseq modules

The main function is to convert individual VCF files (each from a popular somatic mutation caller) to SomaticSeq VCF file. The code to produce the .TSV and .VCF files described in Section 2, for example, would be something like this:

```
# Module is somaticseq/somaticseq/run_somaticseq.py
import somaticseq.somaticseq.run_somaticseq as run_somaticseq

run_somaticseq.runPaired(outdir='/PATH/TO/SomaticSeq', ref='PATH/TO/GRCh38.fa', tbam='/
  PATH/TO/tumor.bwa.bam', nbam='/PATH/TO/normal.bwa.bam', tumor_name='TUMOR', normal_name
  ='NORMAL', truth_snv=None, truth_indel=None, classifier_snv=None, classifier_indel=None,
    pass_threshold=0.5, lowqual_threshold=0.1, hom_threshold=0.85, het_threshold=0.01,
  dbsnp='/PATH/TO/dbSNP_138.hg38.vcf.vcf', cosmic='/PATH/TO/COSMIC.v85.vcf', inclusion='/
  PATH/TO/Exon_Capture.bed', exclusion='/PATH/TO/ignore.bed', mutect=None, indelocator=
  None, mutect2='/PATH/TO/MuTect2.vcf', varscan_snv=None, varscan_indel=None, jsm=None,
  sniper=None, vardict='/PATH/TO/VarDict.vcf', muse='/PATH/TO/MuSE.vcf', lofreq_snv='/PATH
  /TO/LoFreq.snv.vcf.gz', lofreq_indel='/PATH/TO/LoFreq.indel.vcf.gz', scalpel=None,
  strelka_snv='/PATH/TO/Strelka/results/variants/somatic_ssnv.vcf.gz', strelka_indel='/
  PATH/TO/Strelka/results/variants/somatic_sindel.vcf.gz', tnscope=None, min_mq=1, min_bq
  =5, min_caller=0.5, somaticseq_train=False, ensembleOutPrefix='Ensemble.',
  consensusOutPrefix='Consensus.', classifiedOutPrefix='SSeq.Classified.',
  keep_intermediates=False)
```

ensembleOutPrefix, consensusOutPrefix, and classifiedOutPrefix will dictate the output file names under outdir.
Likewise, the single sample mode to convert various individual VCF outputs would be something like this:

```
import somaticseq.somaticseq.run_somaticseq as run_somaticseq

run_somaticseq.runSingle(outdir='/PATH/TO/SomaticSeq', ref='/PATH/TO/GRCh38.fa', bam='/
  PATH/TO/tumor.bwa.bam', tumor_name='TUMOR', truth_snv=None, truth_indel=None,
  classifier_snv=None, classifier_indel=None, pass_threshold=0.5, lowqual_threshold=0.1,
  hom_threshold=0.85, het_threshold=0.01, dbsnp='/PATH/TO/dbSNP_138.hg38.vcf.vcf', cosmic
  ='/PATH/TO/COSMIC.v85.vcf', inclusion='/PATH/TO/Exon_Capture.bed', exclusion='/PATH/TO/
  ignore.bed', mutect=None, mutect2='/PATH/TO/MuTect2.vcf', varscan=None, vardict='/PATH/
  TO/VarDict.vcf', lofreq='/PATH/TO/LoFreq.vcf', scalpel=None, strelka='/PATH/TO/Strelka.
  vcf', min_mq=1, min_bq=5, min_caller=0.5, somaticseq_train=False, ensembleOutPrefix='
  Ensemble.', consensusOutPrefix='Consensus.', classifiedOutPrefix='SSeq.Classified.',
  keep_intermediates=False)
```

Another useful module is the command to extract SomaticSeq features for variants in *any* VCF file, and output the results to a TSV file. The following function requires both tumor and normal BAM files, and the reference genome. COSMIC, dbSNP, etc. are optional. None for any null inputs. min_mq = 0 for this purpose. This is a filter to only output variants that has been called by a minimum number of tools (which you may specify as VCF inputs such as mutect, varscan, etc.)

```
import somaticseq.somaticseq.somatic_vcf2tsv as somatic_vcf2tsv

somatic_vcf2tsv.vcf2tsv(is_vcf='/PATH/TO/variants.vcf', is_bed=None, is_pos=None, nbam_fn
  ='/PATH/TO/normal.bam', tbam_fn='/PATH/TO/tumor.bam', truth=None, cosmic='/PATH/TO/
  COSMIC.v85.vcf', dbsnp='/PATH/TO/dbSNP_138.hg38.vcf.vcf', mutect=None, varscan=None, jsm
  =None, sniper=None, vardict=None, muse=None, lofreq=None, scalpel=None, strelka=None,
  tnscope=None, dedup=True, min_mq=1, min_bq=5, min_caller=0, ref_fa='/PATH/TO/GRCh38.fa',
    p_scale=None, outfile='/PATH/TO/SomaticSeq.FeaturesExtracted.tsv')
```

You may also extract sequencing info for any VCF file if you just have one bam file

```
import somaticseq.somaticseq.single_sample_vcf2tsv as single_sample_vcf2tsv

single_sample_vcf2tsv.vcf2tsv(is_vcf='/PATH/TO/variants.vcf', is_bed=None, is_pos=None,
  bam_fn='/PATH/TO/tumor.bam', truth=None, cosmic='/PATH/TO/COSMIC.v85.vcf', dbsnp='/PATH/
```

```
    TO/dbSNP_138.hg38.vcf.vcf', mutect=None, varscan=None, vardict=None, muse=None, lofreq=
    None, scalpel=None, strelka=None, dedup=True, min_mq=1, min_bq=5, min_caller=0, ref_fa
    ='/PATH/TO/GRCh38.fa', p_scale=None, outfile='/PATH/TO/SomaticSeq.FeaturesExtracted.tsv
    ')
```

Both *somaticseq/somaticseq/somatic_vcf2tsv.py* and *somaticseq/somaticseq/single_sample_vcf2tsv.py* may also be run as standalone scripts. Invoke the script with -h to learn their usages.

4 The step-by-step SomaticSeq Workflow

We'll describe the workflow here.

4.1 Apply inclusion and exclusion regions

This step may be needed for model training. BEDTools is invoked by SomaticSeq. An inclusion region means we will only use calls inside these regions. An exclusion region means we do not care about calls inside this region. DREAM Challenge had exclusion regions, e.g., blacklisted regions, etc. It is also a routine used to parallelize the process by splitting large regions into equal-sized (in terms of number of pairs) regions, so that they can be processed in parallel.

4.2 Combine the call sets

We use *vcfModifier/getUniqueVcfPositions.py* and *bedtools sort* to combine the VCF files from different callers. For each caller output, intermediate VCF file(s) may be created to separate the SNVs and INDELs calls, and also remove some REJECT calls to reduce file sizes.
The following scripts are used to modify original VCF outputs.

```
vcfModifier/modify_JointSNVMix2.py
vcfModifier/modify_MuTect2.py
vcfModifier/modify_MuTect.py
vcfModifier/modify_SomaticSniper.py
vcfModifier/modify_ssMuTect2.py
vcfModifier/modify_ssStrelka.py
vcfModifier/modify_Strelka.py
vcfModifier/modify_VarDict.py
vcfModifier/modify_VarScan2.py
```

modify_ssTOOL.py denotes it's for single-sample mode.
JointSNVMix2 does not output VCF files. In our own workflow, we convert its output into a basic VCF file with an 2 awk one-liners, which you may see at *utilities/dockered_pipelines/mutation_callers/submit_JointSNVMix2.sh*.

```
# To avoid text files on the order of terabytes, this awk one-liner keeps entries where the
    reference is not "N", and the somatic probabilities are at least 0.95.
awk -F "\t" 'NR!=1 && $4!="N" && $10+$11>=0.95'

# This awk one-liner converts the text file into a basic VCF file
awk -F "\t" '{print $1 "\t" $2 "\t.\t" $3 "\t" $4 "\t.\t.\tAAAB=" $10 ";AABB=" $11 "\tRD:AD\
    t" $5 ":" $6 "\t" $7 ":" $8}'

## The actual commands we've used in our workflow:
echo -e '##fileformat=VCFv4.1' > unsorted.vcf
echo -e '##INFO=<ID=AAAB,Number=1,Type=Float,Description="Probability of Joint Genotype AA
    in Normal and AB in Tumor">' >> unsorted.vcf
echo -e '##INFO=<ID=AABB,Number=1,Type=Float,Description="Probability of Joint Genotype AA
    in Normal and BB in Tumor">' >> unsorted.vcf
echo -e '##FORMAT=<ID=RD,Number=1,Type=Integer,Description="Depth of reference-supporting
    bases (reads1)">' >> unsorted.vcf
```

```
echo -e '##FORMAT=<ID=AD,Number=1,Type=Integer,Description="Depth of variant-supporting
    bases (reads2)">' >> unsorted.vcf
echo -e '#CHROM\tPOS\tID\tREF\tALT\tQUAL\tFILTER\tINFO\tFORMAT\tNORMAL\tTUMOR' >> unsorted.
    vcf

python $PATH/TO/jsm.py classify joint_snv_mix_two genome.GRCh37.fa normal.bam tumor.bam
    trained.parameter.cfg /dev/stdout |\
awk -F "\t" 'NR!=1 && $4!="N" && $10+$11>=0.95' |\
awk -F "\t" '{print $1 "\t" $2 "\t." "\t" $3 "\t" $4 "\t.\t.\tAAAB=" $10 ";AABB=" $11 "\tRD:AD\
    t" $5 ":" $6 "\t" $7 ":" $8}' >> unsorted.vcf
```

4.3 Convert the VCF file into TSV file

This script *somaticseq/somatic_vcf2tsv.py* works for all VCF files (requires input for two BAM files). It extracts information from the BAM files, as well as some individual callers' output VCF files. If the ground truth VCF file is included, a called variant will be annotated as a true positive, and everything will be annotated as a false positive. *somaticseq/single_sample_vcf2tsv.py* is used for single-sample mode.
At the end of this, *Ensemble.sSNV.tsv* and *Ensemble.sINDEL.tsv* are created.
All the options for *somaticseq/somatic_vcf2tsv.py* or *somaticseq/single_sample_vcf2tsv.py* can be found by running.

```
somaticseq/somatic_vcf2tsv.py -h
somaticseq/single_sample_vcf2tsv.py -h
```

Note: Do not worry if Python throws a warning like this.

```
RuntimeWarning: invalid value encountered in double_scalars
  z = (s - expected) / np.sqrt(n1*n2*(n1+n2+1)/12.0)
```

This is to tell you that scipy was attempting some statistical test with empty data. That's usually due to the fact that normal BAM file has no variant reads at that given position. That is why lots of values are NaN for the normal.

4.4 Model Training or Mutation Prediction

You can use Ensemble.sSNV.tsv and Ensemble.sINDEL.tsv files either for model training (provided that their mutation status is annotated with 0 or 1) or mutation prediction. This is done with stochastic boosting algorithm we have implemented in R.
Model training:

```
# Training:
r_scripts/ada_model_builder_ntChange.R Ensemble.sSNV.tsv    Consistent_Mates
    Inconsistent_Mates
r_scripts/ada_model_builder_ntChange.R Ensemble.sINDEL.tsv Strelka_QSS Strelka_TQSS
    Consistent_Mates Inconsistent_Mates
```

Ensemble.sSNV.tsv.ntChange.Classifier.RData and *Ensemble.sINDEL.tsv.ntChange.Classifier.RData* will be created from model training. The arguments after *Ensemble.sSNV.tsv* and *Ensemble.sINDEL.tsv* tells the builder script to ignore those features in training. These features do not improve accuracy in our data sets (mostly WGS data, but they may help other data sets)
Mutation prediction:

```
# Mutation prediction:
r_script/ada_model_predictor.R Ensemble.sSNV.tsv.Classifier.RData    Ensemble.sSNV.tsv
    Trained.sSNV.tsv
r_script/ada_model_predictor.R Ensemble.sINDEL.tsv.Classifier.RData Ensemble.sINDEL.tsv
    Trained.sINDEL.tsv
```

After mutation prediction, if you feel like it, you may convert Trained.sSNV.tsv and Trained.sINDEL.tsv into VCF files. Use -tools to list ONLY the individual tools used to have appropriately annotated VCF files. Accepted tools are MuTect2/MuTect/Indelocator, VarScan2, JointSNVMix2, SomaticSniper, VarDict, MuSE, LoFreq, Scalpel, Strelka, and/or TNscope. To list a tool without having run it, the VCF will be annotated as if the tool was run but did not identify that position as a somatic variant, which is probably undesireable.

```
# Probability above 0.7 labeled PASS (-pass 0.7), and between 0.1 and 0.7 labeled LowQual (-
    low 0.1):
# Use -all to include REJECT calls in the VCF file
# Use -phred to convert probability values (between 0 to 1) into Phred scale in the QUAL
    column in the VCF file

somaticseq/SSeq_tsv2vcf.py -tsv Trained.sSNV.tsv   -vcf Trained.sSNV.vcf   -pass 0.7 -low
    0.1 -tools MuTect2 VarScan2 JointSNVMix2 SomaticSniper VarDict MuSE LoFreq Strelka -all
    -phred

somaticseq/SSeq_tsv2vcf.py -tsv Trained.sINDEL.tsv -vcf Trained.sINDEL.vcf -pass 0.7 -low
    0.1 -tools MuTect2 VarScan2 VarDict LoFreq Scalpel Strelka -all -phred
```

5 To run the dockerized somatic mutation callers

For your convenience, we have created a couple of scripts that can generate run script for the dockerized somatic mutation callers.

5.1 Location

- somaticseq/utilities/dockered_pipelines/

5.2 Requirements

- Have internet connection, and able to pull and run docker images from docker.io
- Have cluster management system such as Sun Grid Engine, so that the "qsub" command is valid

5.3 Example commands

5.3.1 Single-threaded Jobs

This is best suited for whole exome sequencing or less.

```
# Example command to submit the run scripts for each of the following somatic mutation
    callers
$PATH/TO/somaticseq/utilities/dockered_pipelines/submit_callers_singleThread.sh \
--normal-bam       /ABSOLUTE/PATH/TO/normal_sample.bam \
--tumor-bam        /ABSOLUTE/PATH/TO/tumor_sample.bam \
--human-reference  /ABSOLUTE/PATH/TO/GRCh38.fa \
--output-dir       /ABSOLUTE/PATH/TO/RESULTS \
--dbsnp            /ABSOLUTE/PATH/TO/dbSNP.GRCh38.vcf \
--somaticseq-dir   /ABSOLUTE/PATH/TO/SomaticSeq \
--action           echo \
```

```
--mutect2 --somaticsniper --vardict --muse --lofreq --scalpel --strelka --somaticseq
```

The command shown above will create scripts for MuTect2, SomaticSniper, VarDict, MuSE, LoFreq, Scalpel, and Strelka. Then, it will create the SomaticSeq script that merges those 7 callers. This command defaults to majority-vote consensus.
Since it's --aciton echo, it will echo the mutation caller scripts locations, but these scripts will not be run. If you do *--action qsub* instead, then those mutation caller scripts will be qsub'ed. You'll still need to mantually run/submit the SomaticSeq script after all the caller jobs are done.

5.3.2 Multi-threaded Jobs

This is best suited for whole genome sequencing. This is same as above, except it will create 36 equal-size regions in 36 bed files, and parallelize the jobs into 36 regions.

```
# Submitting mutation caller jobs by splitting each job into 36 even regions.
$PATH/TO/somaticseq/utilities/dockered_pipelines/submit_callers_multiThreads.sh \
--normal-bam       /ABSOLUTE/PATH/TO/normal_sample.bam \
--tumor-bam        /ABSOLUTE/PATH/TO/tumor_sample.bam \
--human-reference  /ABSOLUTE/PATH/TO/GRCh38.fa \
--output-dir       /ABSOLUTE/PATH/TO/RESULTS \
--dbsnp            /ABSOLUTE/PATH/TO/dbSNP.GRCh38.vcf \
--threads          36 \
--action           echo \
--mutect2 --somaticsniper --vardict --muse --lofreq --scalpel --strelka --somaticseq
```

5.3.3 SomaticSeq Training

Two classifiers will be created (*.RData files), one for SNV and one for INDEL.

```
# Submitting mutation caller jobs by splitting each job into 36 even regions.
$PATH/TO/somaticseq/utilities/dockered_pipelines/submit_callers_singleThread.sh \
--normal-bam       /ABSOLUTE/PATH/TO/normal_sample.bam \
--tumor-bam        /ABSOLUTE/PATH/TO/tumor_sample.bam \
--truth-snv        /ABSOLUTE/PATH/TO/snvTruth.vcf \
--truth-indel      /ABSOLUTE/PATH/TO/indelTruth.vcf \
--human-reference  /ABSOLUTE/PATH/TO/GRCh38.fa \
--output-dir       /ABSOLUTE/PATH/TO/RESULTS \
--dbsnp            /ABSOLUTE/PATH/TO/dbSNP.GRCh38.vcf \
--somaticseq-dir   /ABSOLUTE/PATH/TO/SomaticSeq \
--action           echo \
--mutect2 --somaticsniper --vardict --muse --lofreq --scalpel --strelka --somaticseq --
    somaticseq-train
```

Notice the command includes –truth-snv and –truth-indel, and invokes somaticseq-train.
For multi-threaded job, you should not invoke somaticseq-train. Instead, you should combine all the *Ensemble.sSNV.tsv* and *Ensemble.sINDEL.tsv* files (separately), and then train on the combined files.

5.3.4 SomaticSeq Prediction

```
# Submitting mutation caller jobs by splitting each job into 36 even regions.
$PATH/TO/somaticseq/utilities/dockered_pipelines/submit_callers_singleThread.sh \
--normal-bam       /ABSOLUTE/PATH/TO/normal_sample.bam \
--tumor-bam        /ABSOLUTE/PATH/TO/tumor_sample.bam \
--classifier-snv   /ABSOLUTE/PATH/TO/Ensemble.sSNV.tsv.ntChange.Classifier.RData \
--classifier-indel /ABSOLUTE/PATH/TO/Ensemble.sINDEL.tsv.ntChange.Classifier.RData \
--human-reference  /ABSOLUTE/PATH/TO/GRCh38.fa \
--output-dir       /ABSOLUTE/PATH/TO/RESULTS \
```

```
--dbsnp              /ABSOLUTE/PATH/TO/dbSNP.GRCh38.vcf \
--somaticseq-dir     /ABSOLUTE/PATH/TO/SomaticSeq \
--action             echo \
--mutect2 --somaticsniper --vardict --muse --lofreq --scalpel --strelka --somaticseq
```

Notice the command includes –classifier-snv and –classifier-indel.

5.3.5 Parameters

```
--normal-bam                    /ABSOLUTE/PATH/TO/normal_sample.bam (Required)
--tumor-bam                     /ABSOLUTE/PATH/TO/tumor_sample.bam  (Required)
--human-reference               /ABSOLUTE/PATH/TO/human_reference.fa (Required)
--dbsnp                         /ABSOLUTE/PATH/TO/dbsnp.vcf (Required for MuSE and LoFreq)
--cosmic                        /ABSOLUTE/PATH/TO/cosmic.vcf (Optional)
--selector                      /ABSOLUTE/PATH/TO/Capture_region.bed (Optional. Will assume
    whole genome from the .fai file without it.)
--exclude                       /ABSOLUTE/PATH/TO/Blacklist_region.bed (Optional)
--min-af                        (Optional. The minimum VAF cutoff for VarDict and VarScan2.
    Defaults are 0.10 for VarScan2 and 0.05 for VarDict).
--action                        qsub (Optional: the command preceding the .cmd scripts.
    Default is echo)
--threads                       36 (Optional for multiThreads and invalid for singleThread:
    evenly split the genome into 36 BED files. Default = 12).
--mutect2                       (Optional flag to invoke MuTect2)
--varscan2                      (Optional flag to invoke VarScan2)
--jointsnvmix2                  (Optional flag to invoke JointSNVMix2)
--somaticsniper                 (Optional flag to invoke SomaticSniper)
--vardict                       (Optional flag to invoke VarDict)
--muse                          (Optional flag to invoke MuSE)
--lofreq                        (Optional flag to invoke LoFreq)
--scalpel                       (Optional flag to invoke Scalpel)
--strelka                       (Optional flag to invoke Strelka)
--somaticseq                    (Optional flag to invoke SomaticSeq. This script always be
    echo'ed, as it should not be submitted until all the callers above complete).
--output-dir                    /ABSOLUTE/PATH/TO/OUTPUT_DIRECTORY (Required)
--somaticseq-dir                SomaticSeq_Output_Directory (Optional. The directory name of
    the SomaticSeq output. Default = SomaticSeq).
--somaticseq-train              (Optional flag to invoke SomaticSeq to produce classifiers if
    ground truth VCF files are provided. Only recommended in singleThread mode, because
    otherwise it's better to combine the output TSV files first, and then train classifiers
    .)
--somaticseq-action             (Optional. What to do with the somaticseq.cmd. Default is echo
    . Only do "qsub" if you have already completed all the mutation callers, but want to run
     SomaticSeq at a different setting.)
--classifier-snv                Trained_sSNV_Classifier.RData (Optional if there is a
    classifer you want to use)
--classifier-indel              Trained_sINDEL_Classifier.RData (Optional if there is a
    classifer you want to use)
--truth-snv                     sSNV_ground_truth.vcf (Optional if there is a ground truth,
    and everything else will be labeled false positive)
--truth-indel                   sINDEL_ground_truth.vcf (Optional if there is a ground truth,
    and everything else will be labeled false positive)
--exome                         (Optional flag for Strelka)
--scalpel-two-pass              (Optional parameter for Scalpel. Default = false.)
--mutect2-arguments             (Extra parameters to pass onto Mutect2, e.g., --mutect2-
    arguments '--initial_tumor_lod 3.0 --log_somatic_prior -5.0 --min_base_quality_score
    20')
--mutect2-filter-arguments      (Extra parameters to pass onto FilterMutectCalls)
--varscan-arguments             (Extra parameters to pass onto VarScan2)
--varscan-pileup-arguments      (Extra parameters to pass onto samtools mpileup that creates
    pileup files for VarScan)
--jsm-train-arguments           (Extra parameters to pass onto JointSNVMix2's train command)
--jsm-classify-arguments        (Extra parameters to pass onto JointSNVMix2's classify command
    )
--somaticsniper-arguments       (Extra parameters to pass onto SomaticSniper)
```

```
--vardict-arguments              (Extra parameters to pass onto VarDict)
--muse-arguments                 (Extra parameters to pass onto MuSE)
--lofreq-arguments               (Extra parameters to pass onto LoFreq)
--scalpel-discovery-arguments    (Extra parameters to pass onto Scalpel's discovery command)
--scalpel-export-arguments       (Extra parameters to pass onto Scalpel's export command)
--strelka-config-arguments       (Extra parameters to pass onto Strelka's config command)
--strelka-run-arguments          (Extra parameters to pass onto Strekla's run command)
--somaticseq-arguments           (Extra parameters to pass onto SomaticSeq.Wrapper.sh)
```

5.3.6 What does the single-threaded command do

- For each flag such as --mutect2, --jointsnvmix2, ..., --strelka, a run script ending with .cmd will be created in /ABSOLUTE/PATH/TO/RESULTS/logs. By default, these .cmd scripts will only be created, and their file path will be printed on screen. However, if you do "--action qsub", then these scripts will be submitted via the qsub command. The default action is "echo."
  - Each of these .cmd script correspond to a mutation caller you specified. They all use docker images.
  - We may improve their functionalities in the future to allow more tunable parameters. For the initial releases, POC and reproducibility take precedence.
- If you do "--somaticseq," the somaticseq script will be created in /ABSOLUTE/PATH/TO/RESULTS/SomaticSeq/logs. However, it will not be submitted until you manually do so after each of these mutation callers is finished running.
  - In the future, we may create more sophisticated solution that will automatically solves these dependencies. For the initial release, we'll focus on stability and reproducibility.
- Due to the way those run scripts are written, the Sun Grid Engine's standard error log will record the time the task completes (i.e., Done at 2017/10/30 29:03:02), and it will only do so when the task is completed with an exit code of 0. It can be a quick way to check if a task is done, by looking at the final line of the standard error log file.

5.3.7 What does the multi-threaded command do

It's very similar to the single-threaded WES solution, except the job will be split evenly based on genomic lengths.

- If you specified "--threads 36," then 36 BED files will be created. Each BED file represents 1/36 of the total base pairs in the human genome (obtained from the .fa.fai file, but only including 1, 2, 3, ..., MT, or chr1, chr2, ..., chrM contigs). They are named 1.bed, 2.bed, ..., 36.bed, and will be created into /ABSOLUTE/PATH/TO/RESULTS/1, /ABSOLUTE/PATH/TO/RESULTS/2, ..., /ABSOLUTE/PATH/TO/RESULTS/36. You may, of course, specify any number. The default is 12.
- For each mutation callers you specify (with the exception of SomaticSniper), a script will be created into /ABSOLUTE/PATH/TO/RESULTS/1/logs, /ABSOLUTE/PATH/TO/RESULTS/2/logs, etc., with partial BAM input. Again, they will be automatically submitted if you do "--action qsub."
- Because SomaticSniper does not support partial BAM input (one would have to manually split the BAMs in order to parallelize SomaticSniper this way), the above mentioned procedure is not applied to SomaticSniper. Instead, a single-threaded script will be created (and potentially qsub'ed) into /ABSOLUTE/PATH/TO/RESULTS/logs.

  - However, because SomaticSniper is by far the fastest tool there, single-thread is doable even for WGS. Even single-threaded SomaticSniper will likely finish before parallelized Scalpel. When I benchmarked the DREAM Challenge Stage 3 by splitting it into 120 regions, Scalpel took 10 hours and 10 minutes to complete 1/120 of the data. SomaticSniper took a little under 5 hours for the whole thing.
  - After SomaticSniper finishes, the result VCF files will be split into each of the /ABSOLUTE/-PATH/TO/RESULTS/1, /ABSOLUTE/PATH/TO/RESULTS/2, etc.

- JointSNVMix2 also does not support partial BAM input. Unlike SomaticSniper, it's slow and takes massive amount of memory. It's not a good idea to run JointSNVMix2 on a WGS data. The only way to do so is to manually split the BAM files and run each separately. We may do so in the future, but JointSNVMix2 is a 5-year old that's no longer being supported, so we probably won't bother.

- Like the single-threaded case, a SomaticSeq run script will also be created for each partition like /ABSOLUTE/PATH/TO/RESULTS/1/SomaticSeq/logs, but will not be submitted until you do so manually.

  - For simplicity, you may wait until all the mutation calling is done, then run a command like

```
find /ABSOLUTE/PATH/TO/RESULTS -name 'somaticseq*.cmd' -exec qsub {} \;
```

6 Use BAMSurgeon to create training data

For your convenience, we have created a couple of wrapper scripts that can generate the run script to create training data using BAMSurgon at somaticseq/utilities/dockered_pipelines/bamSimulator. Descriptions and example commands can be found in the README there.
This pipeline is used to spike in in silico somatic mutations into existing BAM files in order to create a training set for somatic mutations.
After the in silico data are generated, you can use the somatic mutation pipeline on the training data to generate the SomaticSeq classifiers.
Classifiers built on training data work if the training data is similar to the data you want to predict. Ideally, the training data are sequenced on the same platform, same sample prep, and similar depth of coverage as the data of interest.
This method is based on BAMSurgeon, slightly modified into our own fork for some speedups.
The proper citation for BAMSurgeon is Ewing AD, Houlahan KE, Hu Y, et al. Combining tumor genome simulation with crowdsourcing to benchmark somatic single-nucleotide-variant detection. Nat Methods. 2015;12(7):623-30.

6.1 Requirements

- Have internet connection, and able to pull and run docker images from docker.io
- Have cluster management system such as Sun Grid Engine, so that the "qsub" command is valid

6.2 Three scenario to simulate somatic mutations

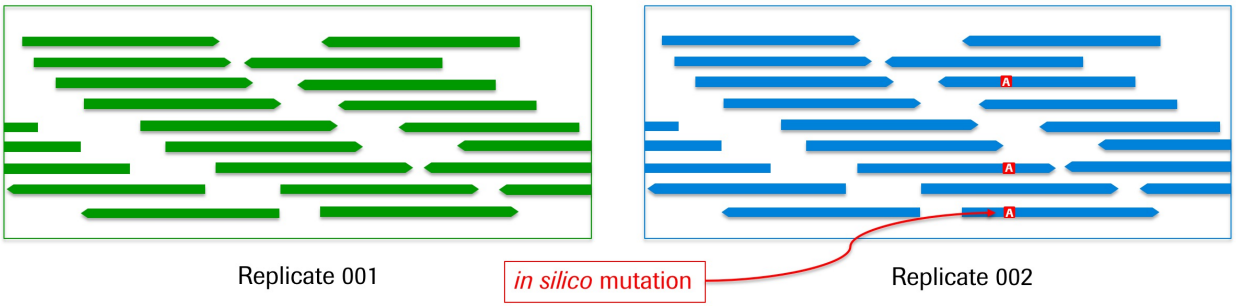
Which scenario to use depend on the data sets available to you.

6.2.1 When you have sequencing replicates of normal samples

This is our approach to define high-confidence somatic mutations in SEQC2 consortium's cancer reference samples, presented here.
In this case, in silico mutations will be spiked into Replicate_002.bam. Since Replicate_002.bam and Replicate_001.bam are otherwise the same sample, any mutations detected that you did not spike in are false positives. The following command is a single-thread example.

```
$PATH/TO/somaticseq/utilities/dockered_pipelines/bamSimulator/BamSimulator_singleThread.sh \
--genome-reference   /ABSOLUTE/PATH/TO/GRCh38.fa \
--tumor-bam-in       /ABSOLUTE/PATH/TO/Replicate_001.bam \
--normal-bam-in      /ABSOLUTE/PATH/TO/Replicate_002.bam \
--tumor-bam-out      syntheticTumor.bam \
--normal-bam-out     syntheticNormal.bam \
--split-proportion   0.5 \
--num-snvs           20000 \
--num-indels         8000 \
--min-vaf            0.0 \
--max-vaf            1.0 \
--left-beta          2 \
--right-beta         5 \
--min-variant-reads  2 \
--output-dir         /ABSOLUTE/PATH/TO/trainingSet \
--action             qsub
```

BamSimulator_*.sh creates semi-simulated tumor-normal pairs out of your input tumor-normal pairs. The "ground truth" of the somatic mutations will be synthetic_snvs.vcf, synthetic_indels.vcf in the output directory.
For multi-thread job (WGS), use BamSimulator_multiThreads.sh instead. See below for additional options and parameters.
A schematic of the BAMSurgeon simulation procedure



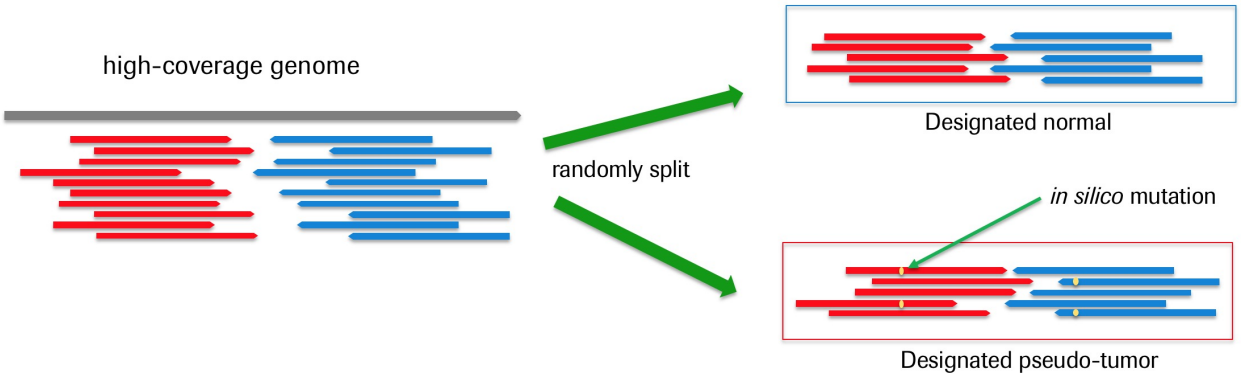
6.2.2 This example mimicks DREAM Challenge

DREAM Somatic Mutation Calling Challenge was an international competition to find algorithms that gave the most accurate performances.
In that case, a high-coverage BAM file is randomly split into two. One of which is designated normal, and the other one is designated tumor where mutations will be spiked in. Like the previous example, any mutations found between the designated tumor and designated normal are false positive, since not only are they from the same sample, but also from the same sequencing run. This example will not capture false positives as a result of run-to-run biases if they exist in your sequencing data. It will, however, still capture artefacts related to sequencing errors, sampling errors, mapping errors, etc.

```
$PATH/TO/somaticseq/utilities/dockered_pipelines/bamSimulator/BamSimulator_multiThreads.sh \
--genome-reference /ABSOLUTE/PATH/TO/GRCh38.fa --tumor-bam-in /ABSOLUTE/PATH/TO/
    highCoverageGenome.bam --tumor-bam-out syntheticTumor.bam --normal-bam-out
    syntheticNormal.bam --split-proportion  0.5 --num-snvs 10000 --num-indels 8000 --num-svs
     1500 --min-vaf 0.0 --max-vaf 1.0 --left-beta 2 --right-beta 5 --min-variant-reads 2 --
```

```
    output-dir /ABSOLUTE/PATH/TO/trainingSet --threads 24 --action qsub --split-bam --indel-
    realign --merge-output-bams
```

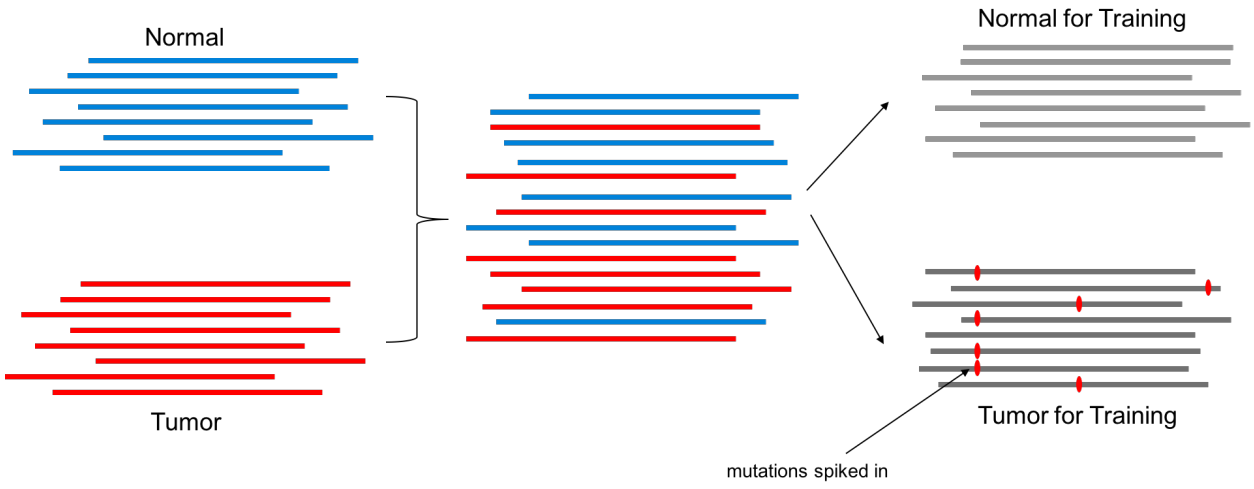
The –split-bem will randomly split the high coverage BAM file into two BAM files, one of which is designated normal and the other one designated tumor for mutation spike in. The –indel-realign is an option that will perform GATK Joint Indel Realignment on the two BAM files. You may or may not invoke it depending on your real data sets. The –merge-output-bams creates another script that will merge the BAM and VCF files region-by-region. It will need to be run manually after all the spike in is done.
A schematic of the DREAM Challenge simulation procedure



6.2.3 Merge and then split the input tumor and normal BAM files

```
$PATH/TO/somaticseq/utilities/dockered_pipelines/bamSimulator/BamSimulator_multiThreads.sh \
--genome-reference /ABSOLUTE/PATH/TO/GRCh38.fa --tumor-bam-in /ABSOLUTE/PATH/TO/Tumor_Sample
    .bam --normal-bam-in /ABSOLUTE/PATH/TO/Normal_Sample.bam --tumor-bam-out syntheticTumor.
    bam --normal-bam-out syntheticNormal.bam --split-proportion 0.5 --num-snvs 30000 --
    num-indels 10000 --num-svs 1500 --min-vaf 0.0 --max-vaf 1.0 --left-beta 2 --right-beta 5
     --min-variant-reads 2 --output-dir /ABSOLUTE/PATH/TO/trainingSet --threads 24 --action
    qsub --merge-bam --split-bam --indel-realign --merge-output-bams
```

The *--merge-bam* will merge the normal and tumor BAM files into a single BAM file. Then, *--split-bem* will randomly split the merged BAM file into two BAM files. One of which is designated normal, and one of which is designated tumor. Synthetic mutations will then be spiked into the designated tumor to create "real" mutations. This is the approach described in our 2017 AACR Abstract.
A schematic of the simulation procedure



6.3 Parameters and Options

```
--genome-reference  /ABSOLUTE/PATH/TO/human_reference.fa (Required)
--selector          /ABSOLUTE/PATH/TO/capture_region.bed (BED file to limit where mutation
    spike in will be attempted)
--tumor-bam-in      Input BAM file (Required)
--normal-bam-in     Input BAM file (Optional, but required if you want to merge it with the
    tumor input)
--tumor-bam-out     Output BAM file for the designated tumor after BAMSurgeon mutation spike
     in
--normal-bam-out    Output BAM file for the designated normal if --split-bam is chosen
--split-proportion  The faction of total reads desginated to the normal. (Defaut = 0.5)
--num-snvs          Number of SNVs to spike into the designated tumor
--num-indels        Number of INDELs to spike into the designated tumor
--num-svs           Number of SVs to spike into the designated tumor (Default = 0)
--min-depth         Minimum depth where spike in can take place
--max-depth         Maximum depth where spike in can take place
--min-vaf           Minimum VAF to simulate
--max-vaf           Maximum VAF to simulate
--left-beta         Left beta of beta distribution for VAF
--right-beta        Right beta of beta distribution for VAF
--min-variant-reads Minimum number of variant-supporting reads for a successful spike in
--output-dir        Output directory
--merge-bam         Flag to merge the tumor and normal bam file input
--split-bam         Flag to split BAM file for tumor and normal
--clean-bam         Flag to go through the BAM file and remove reads where more than 2
    identical read names are present, or reads where its read length and CIGAR string do not
     match. This was necessary for some BAM files downloaded from TCGA. However, a proper
    pair-end BAM file should not have the same read name appearing more than twice. Use this
     only when necessary as it first sorts BAM file by qname, goes through the cleaning
    procedure, then re-sort by coordinates.
--indel-realign     Conduct GATK Joint Indel Realignment on the two output BAM files.
    Instead of syntheticNormal.bam and syntheticTumor.bam, the final BAM files will be
    syntheticNormal.JointRealigned.bam and syntheticTumor.JointRealigned.bam.
--seed              Random seed. Pick any integer for reproducibility purposes.
--threads           Split the BAM files evenly in N regions, then process each (pair) of sub
    -BAM files in parallel.
--action            The command preceding the run script created into /ABSOLUTE/PATH/TO/
    BamSurgeoned_SAMPLES/logs. "qsub" is to submit the script in SGE system. Default = echo
```

6.3.1 –merge-bam / –split-bam / –indel-realign

If you have sequenced replicate normal, that's the best data set for training. You can use one of the normal as normal, and designate the other normal (of the same sample) as tumor. Use *--indel-realign* to invoke GATK IndelRealign.

When you have a normal that's roughly 2X the coverage as your data of choice, you can split that into two halves. One designated as normal, and the other one designated as tumor. That DREAM Challenge's approach. Use *--split-bam --indel-realign* options.

Another approach is to merge the tumor and normal data, and then randomly split them as described above. When you merge the tumor and normal, the real tumor mutations are relegated as germline or noise, so they are considered false positives, because they are supposed to be evenly split into the designated normal. To take this approach, use *--merge-bam --split-bam --indel-realign options*.

Don't use *--indel-realign* if you do not use indel realignment in your alignment pipeline.

In some BAM files, there are reads where read lengths and CIGAR strings don't match. Spike in will fail in these cases, and you'll need to invoke *--clean-bam* to get rid of these problematic reads.

You can control and visualize the shape of target VAF distribution with python command:

```
    import scipy.stats as stats
    import numpy as np
    import matplotlib.pyplot as plt
```

```
    leftBeta, rigthBeta = 2,5
    minAF, maxAF = 0,1
    x = np.linspace(0,1,101)
    y = stats.beta.pdf(x, leftBeta, rigthBeta, loc = minAF, scale = minAF + maxAF)
    _ = plt.plot(x, y)
```

6.4 To create SomaticSeq classifiers

After the mutation simulation jobs are completed, you may create classifiers with the training data with the following command:
See our somatic mutation pipeline for more details.

```
$PATH/TO/somaticseq/utilities/dockered_pipelines/submit_callers_multiThreads.sh \
--output-dir       /ABSOLUTE/PATH/TO/trainingSet/somaticMutationPipeline \
--normal-bam       /ABSOLUTE/PATH/TO/trainingSet/syntheticNormal.bam \
--tumor-bam        /ABSOLUTE/PATH/TO/trainingSet/syntheticTumor.bam \
--human-reference  /ABSOLUTE/PATH/TO/GRCh38.fa \
--dbsnp            /ABSOLUTE/PATH/TO/dbSNP.GRCh38.vcf \
--thread           24 \
--truth-snv        /ABSOLUTE/PATH/TO/trainingSet/synthetic_snvs.vcf \
--truth-indel      /ABSOLUTE/PATH/TO/trainingSet/synthetic_indels.leftAlign.vcf \
--action           echo \
--mutect2 --somaticsniper --vardict --muse --lofreq --strelka --somaticseq
```

7 Release Notes

Make sure training and prediction use the same SomaticSeq version, or at least make sure the different minor version changes do not change the results significantly.

1. Version 1.0

   Version used to generate data in the manuscript and Stage 5 of the ICGC-TCGA DREAM Somatic Mutation Challenge, where SomaticSeq's results were #1 for INDEL and #2 for SNV.

   In the original manuscript, VarDict's var2vcf_somatic.pl script was used to generate VarDict VCFs, and subsequently "-filter somatic" was used for SSeq_merged.vcf2tsv.py. Since then (including DREAM Challenge Stage 5), VarDict recommends var2vcf_paired.pl over var2vcf_somatic.pl, and subsequently "-filter paired" was used for SSeq_merged.vcf2tsv.py. The difference in SomaticSeq results, however, is pretty much negligible.

2. Version 1.1

   Automated the SomaticSeq.Wrapper.sh script for both training and prediction mode. No change to any algorithm.

3. Version 1.2

   Have implemented the following improvement, mostly for indels:

   - SSeq_merged.vcf2tsv.py can now accept pileup files to extract read depth and DP4 (reference forward, reference reverse, alternate forward, and alternate reverse) information (mainly for indels). Previously, that information can only be extracted from SAMtools VCF. Since the SAMtools or HaplotypeCaller generated VCFs hardly contain any indel information, this option improves the indel model. The SomaticSeq.Wrapper.sh script is modified accordingly.
   - Extract mapping quality (MQ) from VarDict output if this information cannot be found in SAMtools VCF (also mostly benefits the indel model).

- Indel length now positive for insertions and negative for deletions, instead of using the absolute value previously.

4. Version 2.0

   - Removed dependencies for SAMtools and HaplotypeCaller during feature extraction. SSeq_merged.vcf2tsv.py extracts those information (plus more) directly from BAM files.
   - Allow not only VCF file, but also BED file or a list of chromosome coordinate as input format for SSeq_merged.vcf2tsv.py, i.e., use -mybed or -mypos instead of -myvcf.
   - Instead of a separate step to annotate ground truth, that can be done directly by SSeq_merged.vcf2tsv.py by supplying the ground truth VCF via -truth.
   - SSeq_merged.vcf2tsv.py can annotate dbSNP and COSMIC information directly if BED file or a list of chromosome coordinates are used as input in lieu of an annotated VCF file.
   - Consolidated feature sets, e.g., removed some redunda Fixed a bug: if JointSNVMix2 is not included, the values should be "NaN" instead of 0's. This is to keep consistency with how we handle all other callersnt feature sets coming from different resources.

5. Version 2.0.2

   - Incorporated LoFreq.
   - Used getopt to replace getopts in the SomaticSeq.Wrapper.sh script to allow long options.

6. Version 2.1.2

   - Properly handle cases when multiple ALT's are calls in the same position. The VCF files can either contain multiple calls in the ALT column (i.e., A,G), or have multiple lines corresponding to the same position (one line for each variant call). Some functions were significantly rewritten to allow this.
   - Incorporated Scalpel.
   - Deprecated HaplotypeCaller and SAMTools dependencies completely as far as feature generation is concerned.
   - The Wrapper script removed SnpSift/SnpEff dependencies. Those information can be directly obtained during the SSeq_merged.vcf2tsv.py step. Also removed some additional legacy steps that has become useless since v2 (i.e., score_Somatic.Variants.py). Added a step to check the correctness of the input. The v2.1 and 2.1.1 had some typos in the wrapper script, so only describing v2.1.2 here.

7. Version 2.2

   - Added MuTect2 support.

8. Version 2.2.1

   - InDel_3bp now stands for indel counts within 3 bps of the variant site, instead of exactly 3 bps from the variant site as it was previously (likewise for InDel_2bp).
   - Collapse MQ0 (mapping quality of 0) reads supporting reference/variant reads into a single metric of MQ0 reads (i.e., tBAM_MQ0 and nBAM_MQ0). From experience, the number of MQ0 reads is at least equally predictive of false positive calls, rather than distinguishing if those MQ0 reads support reference or variant.
   - Obtain SOR (Somatic Odds Ratio) from BAM files instead of VarDict's VCF file.
   - Fixed a typo in the SomaticSeq.Wrapper.sh script that did not handle inclusion region correctly.

9. Version 2.2.2

    - Got around an occasional unexplained issue in then ada package were the SOR is sometimes categorized as type, by forcing it to be numeric.
    - Defaults PASS score from 0.7 to 0.5, and make them tunable in the SomaticSeq.Wrapper.sh script (--pass-threshold and --lowqual-threshold).

10. Version 2.2.3

    - Incorporated Strelka2 since it's now GPLv3.
    - Added another R script (ada_model_builder_ntChange.R) that uses nucleotide substitution pattern as a feature. Limited experiences have shown us that it improves the accuracy, but it's not heavily tested yet.
    - If a COSMIC site is labeled SNP in the COSMIC VCF file, if_cosmic and CNT will be labeled as 0. The COSMIC ID will still appear in the ID column. This will not change any results because both of those features are turned off in the training R script.
    - Fixed a bug: if JointSNVMix2 is not included, the values should be "NaN" instead of 0's. This is to keep consistency with how we handle all other callers.

11. Version 2.2.4

    - Resolved a bug in v2.2.3 where the VCF files of Strelka INDEL and Scalpel clash on GATK CombineVariants, by outputting a temporary VCF file for Strelka INDEL without the sample columns.
    - Caller classification: consider if_Scalpel = 1 only if there is a SOMATIC flag in its INFO.

12. Version 2.2.5

    - Added a dockerfile. Docker repo at https://hub.docker.com/r/lethalfang/somaticseq/.
    - Ability to use vcfsort.pl instead of GATK CombineVariants to merge VCF files.

13. Version 2.3.0

    - Moved some scripts to the utilities directory to clean up the clutter.
    - Added the split_Bed_into_equal_regions.py to utilities, which will split a input BED file into multiple BED files of equal size. This is to be used to parallelize large WGS jobs.
    - Made compatible with MuTect2 from GATK4.
    - Removed long options for the SomaticSeq.Wrapper.sh script because it's more readable this way.
    - Added a script to add "GT" field to Strelka's VCF output before merging it with other VCF files. That was what caused GATK CombineVariants errors mentioned in v2.2.4's release notes.
    - Added a bunch of scripts at utilities/dockered_pipelines that can be used to submit (requiring Sun Grid Engine or equivalent) dockerized pipeline to a computing cluster.

14. Version 2.3.1

    - Improve the automated run script generator at utilities/dockered_pipelines.
    - No change to SomaticSeq algorithm

15. Version 2.3.2

    - Added run script generators for dockerized BAMSurgeon pipelines at utilities/dockered_pipelines/bamSurgeon

- Added an error message to r_scripts/ada_model_builder_ntChange.R when TrueVariants_or_False don't have both 0's and 1's. Other than this warning message change, no other change to SomaticSeq algorithm.

16. Version 2.4.0

    - Restructured the utilities scripts.
    - Added the utilities/filter_SomaticSeq_VCF.py script that "demotes" PASS calls to LowQual based on a set of tunable hard filters.
    - BamSurgeon scripts invokes modified BamSurgeon script that splits a BAM file without the need to sort by read name. This works if the BAM files have proper read names, i.e., 2 and only 2 identical read names for each paired-end reads.
    - No change to SomaticSeq algorithm

17. Version 2.4.1

    - Updated some docker job scripts.
    - Added a script that converts some items in the VCF's INFO field into the sample field, to precipitate the need to merge multiple VCF files into a single multi-sample VCF, i.e., utilities/reformat_VCF2SEQC2.py.
    - No change to SomaticSeq algorithm

18. Version 2.5.0

    - In modify_VJSD.py, get rid of VarDict's END tag (in single sample mode) because it causes problem with GATK CombineVariants.
    - Added limited single-sample support, i.e., ssSomaticSeq.Wrapper.sh is the wrapper script. singleSample_callers_singleThread.sh is the wrapper script to submit single-sample mutation caller scripts.
    - Added run scripts for read alignments and post-alignment processing, i.e,. FASTQ $\rightarrow$ BAM, at utilities/dockered_pipelines/alignments.
    - Fixed a bug where the last two CD4 numbers were both alternate concordant reads in the output VCF file, when the last number should've been alternate discordant reads.
    - Changed the output file names from Trained.s(SNV|INDEL).vcf and Untrained.s(SNV|INDEL).vcf to SSeq.Classified.s(SNV|INDE).vcf and Consensus.s(SNV|INDEL).vcf. No change to the actual tumor-normal SomaticSeq algorithm.
    - Added utilities/modify_VarDict.py to VarDict's "complex" variant calls (e.g., GCA¿TAC) into SNVs when possible.
    - Modified r_scripts/ada_model_builder_ntChange.R to allow you to ignore certain features, e.g., r_scripts/ada_model_builder_ntChange.R Training_Data.tsv nBAM_REF_BQ tBAM_REF_BQ SiteHomopolymer_Length ...

      Everything after the input file are features to be ignored during training.

      Also added r_scripts/ada_cross_validation.R.

19. Version 2.5.1

    - Additional passable parameters options to pass extra parameters to somatic mutation callers. Fixed a bug where the "two-pass" parameter is not passed onto Scalpel in multiThreads scripts.
    - Ignore Strelka_QSS and Strelka_TQSS for indel training in the SomaticSeq.Wrapper.sh script.

20. Version 2.5.2

- Ported some pipeline scripts to singularities at utilities/singularities.

21. Version 2.6.0

    - VarScan2_Score is no longer extracted from VarScan's output. Rather, it's now calculated directly using Fisher's Exact Test, which reproduces VarScan's output, but will have a real value when VarScan2 does not output a particular variant.
    - Incorporate TNscope's output VCF into SomaticSeq, but did not incorporate TNscope caller into the dockerized workflow because we don't have distribution license.

22. Version 2.6.1

    - Optimized memory for singularity scripts.
    - Updated utilities/bamQC.py and added utilities/trimSoftClippedReads.py (removed soft-clipped bases on soft-clipped reads)
    - Added some docker scripts at utilities/dockered_pipelines/QC

23. Version 2.7.0

    - Added another feature: consistent/inconsistent calls for paired reads if the position is covered by both forward and reverse reads. However, they're excluded as training features in SomaticSeq.Wrapper.sh script for the time being.
    - Change non-GCTA characters to N in VarDict.vcf file to make it conform to VCF file specifications.

24. Version 2.7.1

    - Without –gatk $PATH/TO/GenomeAnalysisTK.jar in the SomaticSeq.Wrapper.sh script, it will use utilities/getUniqueVcfPositions.py and utilities/vcfsorter.pl to (in lieu of GATK3 CombineVariants) to combine all the VCF files.
    - Fixed bugs in the docker/singularities scripts where extra arguments for the callers are not correctly passed onto the callers.

25. Version 2.7.2

    - Make compatible with .cram format
    - Fixed a bug where Strelka-only calls are not considered by SomaticSeq.

26. Version 2.8.0

    - The program is now designed to crash if the VCF file(s) are not sorted according to the .fasta reference file.

27. Version 2.8.1

    - Fixed a bug in the ssSomaticSeq.Wrapper.sh script (single-sample mode), where the SNV algorithm weren't looking for SNV VCF files during merging when using utilities/getUniqueVcfPositions.py, causing empty SNV files. For previous commands (invoking –gatk for CombineVariants), the results have never changed.

28. Version 3.0.0

    Refactored the codes.

    - The wrapper scripts written in bash script (i.e., SomaticSeq.Wrapper.sh and ssSomaticSeq.Wrapper.sh) are replaced by somaticseq/run_somaticseq.py, though they're still kept for backward-compatibility.
    - Allow parallel processing using somaticseq_parallel.py

8 Contact Us

For suggestions, bug reports, or technical support, please post in https://github.com/bioinform/somaticseq/issues. The developers are alerted when issues are created there. Alternatively, you may also email li_tai.fang@roche.com.