SomaticSeq Documentation

Li Tai Fang / ltfang@gmail.com

February 1, 2021

Contents

1 Introduction

SomaticSeq is a flexible post-somatic-mutation-calling algorithm for improved accuracy. It is compatible with 10+ somatic mutation callers. 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 to distinguish true mutations from false positives from that call set. The mutation callers we have incorporated are MuTect/Indelocator/MuTect2 [1], VarScan2 [2], JointSNVMix [3], SomaticSniper [4], VarDict [5], MuSE [6], LoFreq [7], Scalpel [8], Strelka2 [9], TNscope [10], and Platypus [11]. 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 [12]. 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 script 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

1. SomaticSeq was written in *Python 3* under Linux environment. In addition, Python libraries of *numpy*, *scipy*, and *pysam* are also required.

2. SomaticSeq also uses *BEDTools* [13] to manipulate bed file inputs, i.e., regions to include and/or exclude in the workflow.

3. *R*, as well as *ada* library (if you intend to use this).

4. At its core, SomaticSeq combines and then filters the results of multiple somatic mutation detection algorithms based on many sequencing features. Generally speaking, at least one compatible caller needs to be run to generate a list of mutation candidates for SomaticSeq to evaluate. It is compatible with the following callers: the original MuTect/Indelocator as well as GATK4's Mutect2 [1], VarScan2 [2], JointSNVMix2 [3], SomaticSniper [4], VarDict [5], MuSE [6], LoFreq [7], Scalpel [8], Strelka2 [9], TNscope [10], and Platypus [11].

5. *Docker* [http://www.docker.com] is a container technology that can be used to package softwares and their dependencies in a portable Docker images, which can be used to execute a workflow reproducibly across different platforms and environments. SomaticSeq does not require Docker *per se*, but we have created Docker images of it, along with a number of compatible somatic mutation callers to make life easier for new users. The advantage of using container technologies like Docker, is that one does not necessarily have to create the right software environment with the correct dependencies for every software in a workflow, e.g,. by creating a Docker image for MuTect2, the users can simply use that Docker image for MuTect 2 tasks. Otherwise, they must make sure to have the correct Java

version and others dependencies to run MuTect2, and that those dependencies do not conflict with other software that may need different Java versions.

1.2 Docker images

SomaticSeq and the somatic mutation callers that we routinely use were 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/lethalfang/lofreq
- Scalpel: https://hub.docker.com/r/lethalfang/scalpel
- Strelka2: https://hub.docker.com/r/lethalfang/strelka

2 Download and install SomaticSeq

Source code of SomaticSeq is available via Github repository in BSD 2-Clause open source license: https://github.com/bioinform/somaticseq. The latest source code can be cloned by the git command:

```
git clone https://github.com/bioinform/somaticseq.git
```

To install SomaticSeq, then the core script *somaticseq_parallel.py* will be in your path.

```
./setup.py install
```

3 How to run SomaticSeq

The *somaticseq_parallel.py* module calls a series of programs and procedures **after** you have run your individual somatic mutation callers. 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

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

# Parameters for single-sample mode
somaticseq_parallel.py single --help
```

3.1 SomaticSeq Training Mode

To create SomaticSeq classifiers, you need a VCF file containing true SNVs and a VCF file containing true INDELs, and invoke the training node with *--somaticseq-train* flag. 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 and will be labeled such. Every mutation call not in the truth VCF files is assumed to be false positives and will be labeled as such (as long as the genomic coordiante is in inclusion region and not in exclusion region if those regions are provided).

All the VCF files from individual callers are optional, but you need at least one or there will be nothing to do. All 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.

```
somaticseq_parallel.py \
--output-directory      OUTPUT_DIR \
--genome-reference      GRCh38.fa \
--inclusion-region      genome.bed \
--exclusion-region      blacklist.bed \
--truth-snv             truePositives.snv.vcf \
--truth-indel           truePositives.indel.vcf \
--threads               28 \
--somaticseq-algorithm  xgboost \
--somaticseq-train      \
paired                  \
--tumor-bam-file        tumor.bam \
--normal-bam-file       matched_normal.bam \
--mutect2-vcf           MuTect2.vcf \
--vardict-vcf           VarDict.vcf \
--muse-vcf              MuSE.vcf \
--strelka-snv           Strelka/results/variants/somatic.snvs.vcf.gz \
--strelka-indel         Strelka/results/variants/somatic.indels.vcf.gz
```

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.

3.2 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.xgb.v3.x.x.classifier \
--classifier-indel      Ensemble.sINDEL.tsv.xgb.v3.x.x.classifier \
--output-directory      OUTPUT_DIR \
--genome-reference      GRCh38.fa \
--inclusion-region      genome.bed \
--exclusion-region      blacklist.bed \
--somaticseq-algorithm  xgboost \
--threads               12 \
paired                  \
--tumor-bam-file        tumor.bam \
--normal-bam-file       matched_normal.bam \
--mutect2-vcf           MuTect2/variants.vcf \
--vardict-vcf           VarDict/variants.vcf \
--muse-vcf              MuSE/variants.snp.vcf \
--strelka-snv           Strelka/variants.snv.vcf \
--strelka-indel         Strelka/variants.indel.vcf
```

3.3 SomaticSeq Consensus Mode

Same as the commands previously, but without including the classifiers (.RData files) or invoking *–somaticseq-train*. Without those information, SomaticSeq will forgo machine learning, and fall back into a simple majority vote. The following is an example:

```
# The *.RData files are trained classifier from the training mode.
somaticseq_parallel.py \
--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 \
--vardict-vcf       VarDict/variants.vcf \
--muse-vcf          MuSE/variants.snp.vcf \
--strelka-snv       Strelka/variants.snv.vcf \
--strelka-indel     Strelka/variants.indel.vcf
```

4 SomaticSeq as a Python library

Section 3 described how to use SomaticSeq as a software. It is also possible to treat SomaticSeq as a python library extension for your own software. So here we describe in detail the procedures and functions that make up SomaticSeq.

4.1 Module: somaticseq_parallel.py

The *somaticseq_parallel.py* script simply calls for *somaticseq/run_somaticseq.py* module (Sec. 4.2), and parallelize the runs by splitting the input BED file into a number of equal-sized (in terms of total base pairs) regions. The BED file splitting is achieved by *somaticseq/utilities/split_Bed_into_equal_regions.py* (Sec. 4.6.1).

4.2 Module: somaticseq/run_somaticseq.py

The core module for SomaticSeq is *somaticseq/run_somaticseq.py*. It converts individual VCF files from somatic mutation caller(s) into SomaticSeq TSV and VCF files. There are two main functions in the module, *runPaired* and *runSingle*. Depending on the mode, either of them can be called. For example:

```
# Module is located 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, platypus=None, min_mq=1, min_bq=5, min_caller=0.5, somaticseq_train=False,
  ensembleOutPrefix='Ensemble.', consensusOutPrefix='Consensus.', classifiedOutPrefix='SSeq.
  Classified.', algo='ada', keep_intermediates=False)
```

The parameters of *ensembleOutPrefix*, *consensusOutPrefix*, and *classifiedOutPrefix* will dictate the output file names under outdir.

We'll briefly describe the procedures of runPaired here. First of all, the *somaticseq/combine_callers.py* module (combinePaired function) will combine all the input VCF files into two minimal VCF files, one for SNVs and one for INDELs, that include each unique variant call. These VCF files serve as input files for the next steps, where features are extracted from each of the variant and then converted to the SomaticSeq TSV files by the *somaticseq/somatic_vcf2tsv.py* module (Sec. 4.3).

If training mode was invoked (Sec. 3.1), SomaticSeq classifiers would be built (Sec. 4.5.1). If prediction mode was invoked, an additional TSV file with prediction scores will be created (Sec. 4.5.2).

Finally, the TSV files will be converted to SomaticSeq VCF file output by somaticseq/SSeq_tsv2vcf.py (Sec. 4.3).

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.', algo='ada', keep_intermediates=False)
```

Parameters:

- truth_snv/truth_indel: if present, then the variants in these VCF files will be considered true positives, and *everything else* will be considered false positive. If None, then nothing with regard to true positive or false positive will be annotated.
- classifier_snv/classifier_indel: if present, then SomaticSeq prediction will be invoked to create machine learning classified VCF files. if None, only majority-vote consensus VCF files will be created.
- inclusion: bed file so only variants in it will be considered (requires BEDTools on execution path)
- exclusion: bed file so variants in it will be tossed out (requires BEDTools on the execution path)
- mutect/mutect2/varscan/jsm/vardict/muse/lofreq/strelka/scalpel/tnscope: output VCF files from the callers. If None, then it assumes that tool was not used.
- min_caller: only output variants if at least N number of callers have called it. Since some LowQual calls are considered 0.5, an input of 0.5 tells the function to also return variants even if it's only been called as a "LowQual" by a tool. However, it will still filter out variants that's only been "REJECTED" by a caller.

  somaticseq_train: if True, and also if truth_snv or truth_indel are present, then it will create SomaticSeq classifiers. If False, then will not invoke training mode.

4.3 Module: somaticseq/somatic_vcf2tsv.py and somaticseq/single_sample_vcf2tsv.py

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,
  platypus=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.

Parameters:

- is_vcf: the VCF file serves as the input file, from which every variant will have its sequencing feature extracted from the BAM file(s).
- mutect/varscan/jsm/sniper/vardict/muse/lofreq/scalpel/strelka/tnscope: VCF files from these tools. If present, the function will extract information from these files such as if a variant is called by the tool. If None, everything associated with that tool will be "nan" in the TSV file.

The module can also be run independently on any VCF file to extract SomaticSeq-related features for the variants in the VCF files. To find out how to use them, do this in the command shell:

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

4.4 Module: somaticseq/SSeq_tsv2vcf.py

This module converts SomaticSeq's TSV file (described in Sec. 4.3) to SomaticSeq VCF files.

```
import somaticseq.somaticseq.SSeq_tsv2vcf as SSeq_tsv2vcf

SSeq_tsv2vcf.tsv2vcf(tsv_fn='/PATH/TO/SomaticSeq.tsv', vcf_fn='/PATH/TO/SomaticSeq.vcf', tools
  =['MuTect2', 'SomaticSniper', 'Strelka'], pass_score=0.5, lowqual_score=0.1, hom_threshold
  =0.85, het_threshold=0.01, single_mode=False, paired_mode=True, normal_sample_name='NORMAL',
  tumor_sample_name='TUMOR', print_reject=True, phred_scaled=True)
```

Parameters:

- tools: A list of tools that were run, can only be selected from MuTect2, MuTect, VarScan2, JointSNVMix2, SomaticSniper, VarDict, MuSE, LoFreq, Scalpel, Strelka, TNscope, and/or Platypus.
- print_reject: if False, will only print PASS and LowQual variants into VCF. If True, will print everything from TSV to VCF.
- phred_scaled: if True, will print Phred-scaled score in QUAL column (if the TSV was produced with SomaticSeq prediction). If False, will print the 0-1 scale. If no SomaticSeq prediction was done, will print 0.

The script can also be run independently.

```
somaticseq/SSeq_tsv2vcf.py -h
```

4.5 Machine learning modules

The training and prediction scripts are written in R.

4.5.1 SomaticSeq Training

ada_model_builder_ntChange.R or somatic_xgboost.py is the script that is called during SomaticSeq pipeline to make classifiers. You can also run them indepedently with labeled Ensemble.sSNV.tsv and Ensemble.sINDEL.tsv files. The command for ada is:

```
# Training:
r_scripts/ada_model_builder_ntChange.R Ensemble.sSNV.tsv
```

For extreme gradient boosting (xgboost) algorithm, the program can be run multi-threaded:

```
# Training:
somatic_xgboost.py train -tsv Ensemble.sSNV.tsv -threads 4
```

4.5.2 Prediction with trained classifiers

ada_model_predictor.R or somatic_xgboost.py is the prediction script. To run it independently, the command for ada is

```
# Mutation prediction:

ada_model_predictor.R Ensemble.sSNV.tsv.Classifier.RData    Ensemble.sSNV.tsv    Predicted.sSNV.tsv
```

If classifier was based on xgboost:

```
# Mutation prediction:
xgboost.py predict -model Ensemble.sSNV.tsv.gz.xgb.v3.x.x.model -tsv Ensemble.sSNV.tsv.gz -out
    SSeq.Predicted.sSNV.tsv
```

4.6 utilities modules

4.6.1 Module: split_Bed_into_equal_regions

Given a .bed or a .fa.fai file, it will split the input region into N number of bed files, such that each bed file has equal-sized regions in them.

4.6.2 Module: lociCounterWithLabels

Given a list of .bed files and a .fa.fai file, it will return a .bed file detailing which regions were contained from which .bed inputs.

```
lociCounterWithLabels.py -fai GRCh38.fa.fai -beds 1.bed 2.bed 3.bed -labels 01 02 03 -out
  overlapping.bed
```

Parameters:

- labels: A list of labels to be written in 4th column of the output bed file. If absent, the 4th column will be populated by the input bed file names.

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

5.3 Example commands

You may run the following command to see all the available options for this command, in either paired (tumor-normal) or single (tumor-only) mode.

```
makeSomaticScripts.py [paired|single] -h
```

5.3.1 Tumor-Normal Mode

```
# Example command to submit the run scripts for each of the following somatic mutation callers
makeSomaticScripts.py paired \
--normal-bam        /ABSOLUTE/PATH/TO/normal_sample.bam \
--tumor-bam         /ABSOLUTE/PATH/TO/tumor_sample.bam \
--genome-reference  /ABSOLUTE/PATH/TO/GRCh38.fa \
--output-directory  /ABSOLUTE/PATH/TO/RESULTS \
--dbsnp-vcf         /ABSOLUTE/PATH/TO/dbSNP.GRCh38.vcf \
--threads           12 \
--run-mutect2 --run-somaticsniper --run-vardict --run-muse --run-lofreq --run-scalpel --run-
    strelka2 --run-somaticseq --run-workflow-locally
```

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.

The *`--threads 12`* will create 12 equal-size regions in 12 bed files, and parallelize the jobs into 12 regions. However, you'll need to combine those 12 seperate results together.

5.3.2 SomaticSeq Training

As things are currently set up, training mode is best run seperately **after** you've run the workflows above, because we don't have a workflow engine to manage and then merge the result of each thread. You may invoke "'–train-somaticseq"' here, but SomaticSeq will train on each thread. Now if you use just a single thread (e.g., "'–threads 1"' is the default), you may train it just fine. In this case, two classifiers will be created (*.RData files), one for SNV and one for INDEL.

```
makeSomaticScripts.py paired \
--normal-bam            /ABSOLUTE/PATH/TO/normal_sample.bam \
--tumor-bam             /ABSOLUTE/PATH/TO/tumor_sample.bam \
--genome-reference      /ABSOLUTE/PATH/TO/GRCh38.fa \
--output-directory      /ABSOLUTE/PATH/TO/RESULTS \
--dbsnp-vcf             /ABSOLUTE/PATH/TO/dbSNP.GRCh38.vcf \
--truth-snv             /ABSOLUTE/PATH/TO/truth.snv.vcf \
--truth-indel           /ABSOLUTE/PATH/TO/truth.indel.vcf \
--somaticseq-algorithm  xgboost \
--train-somaticseq      \
--run-mutect2 --run-somaticsniper --run-vardict --run-muse --run-lofreq --run-scalpel --run-
    strelka2 --run-somaticseq
```

Notice the command includes *–truth-snv* and *–truth-indel*, and invokes somaticseq-train. By default ada will be used for *–somaticseq-algorithm*, but you may invoke xgboost as well.

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.

5.3.3 SomaticSeq Prediction

```
makeSomaticScripts.py paired \
--normal-bam            /ABSOLUTE/PATH/TO/normal_sample.bam \
--tumor-bam             /ABSOLUTE/PATH/TO/tumor_sample.bam \
--genome-reference      /ABSOLUTE/PATH/TO/GRCh38.fa \
--output-directory      /ABSOLUTE/PATH/TO/RESULTS \
--dbsnp-vcf             /ABSOLUTE/PATH/TO/dbSNP.GRCh38.vcf \
--snv-classifier        /ABSOLUTE/PATH/TO/Snv_Classifier.RData \
--indel-classifier      /ABSOLUTE/PATH/TO/Indel_Classifier.RData \
--somaticseq-algorithm xgboost \
--action                echo \
--threads               12 \
--run-mutect2 --run-somaticsniper --run-vardict --run-muse --run-lofreq --run-scalpel --run-
    strelka2 --run-somaticseq
```

Notice the command includes –classifier-snv and –classifier-indel. Make sure the classifier and the –*somaticseq-algorithm* argument matches.

5.3.4 Parameters

```
paired                          Invokes tumor-normal modes. Placed immediately after
    makeSomaticScripts.py.
single                          Invokes tumor-only modes. Placed immediately after
    makeSomaticScripts.py
--normal-bam                    /ABSOLUTE/PATH/TO/normal_sample.bam (Required for paired)
--tumor-bam                     /ABSOLUTE/PATH/TO/tumor_sample.bam (Required for paired)
--bam                           /ABSOLUTE/PATH/TO/tumor_sample.bam (Required for single)
--genome-reference              /ABSOLUTE/PATH/TO/human_reference.fa (Required)
--dbsnp-vcf                     /ABSOLUTE/PATH/TO/dbsnp.vcf (Required: for MuSE and LoFreq)
--cosmic-vcf                    /ABSOLUTE/PATH/TO/cosmic.vcf (Optional)
--inclusion-region              /ABSOLUTE/PATH/TO/Capture_region.bed (Optional. Will assume whole
    genome from the .fai file without it.)
--exclusion-region              /ABSOLUTE/PATH/TO/Blacklist_region.bed (Optional)
--minimum-VAF                   (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 = 1).
--run-mutect2                   (Optional flag to invoke MuTect2)
--run-varscan2                  (Optional flag to invoke VarScan2)
--run-jointsnvmix2              (Optional flag to invoke JointSNVMix2. Not for single.)
--run-somaticsniper             (Optional flag to invoke SomaticSniper. Not for single.)
--run-vardict                   (Optional flag to invoke VarDict)
--run-muse                      (Optional flag to invoke MuSE. Not for single.)
--run-lofreq                    (Optional flag to invoke LoFreq)
--run-scalpel                   (Optional flag to invoke Scalpel)
--run-strelka                   (Optional flag to invoke Strelka)
--run-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-directory              /ABSOLUTE/PATH/TO/OUTPUT_DIRECTORY (Required)
--somaticseq-directory          SomaticSeq_Output_Directory (Optional. The directory name of the
    SomaticSeq output. Default = SomaticSeq).
--train-somaticseq              (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.)
--snv-classifier                Trained_sSNV_Classifier.RData (Optional if there is a classifer you
    want to use)
--indel-classifier              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)
--somaticseq-algorithm         Default is ada, but you may also use xgboost.
```

5.3.5 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.

5.3.6 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.

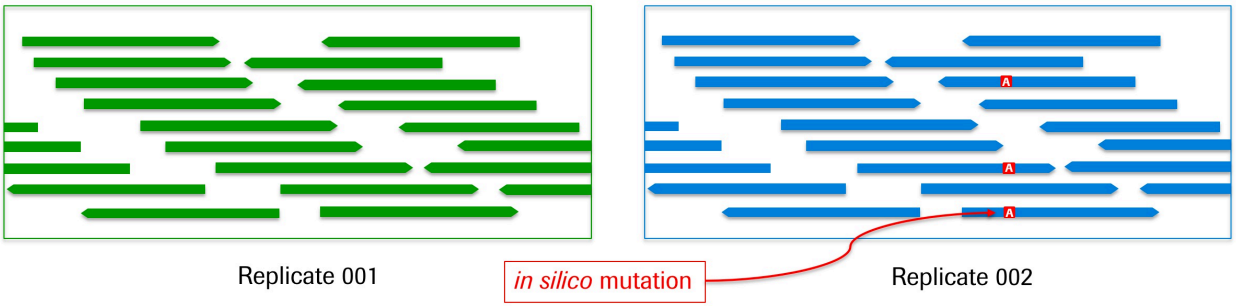
In this case, in silico mutations will be spiked into Replicate_002.bam. Since Replicate_002.bam and Replicate_001.bam are otherwise the same sample, any mutations detected that you did not spike in are false positives. The following command is a single-thread example.

```
$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

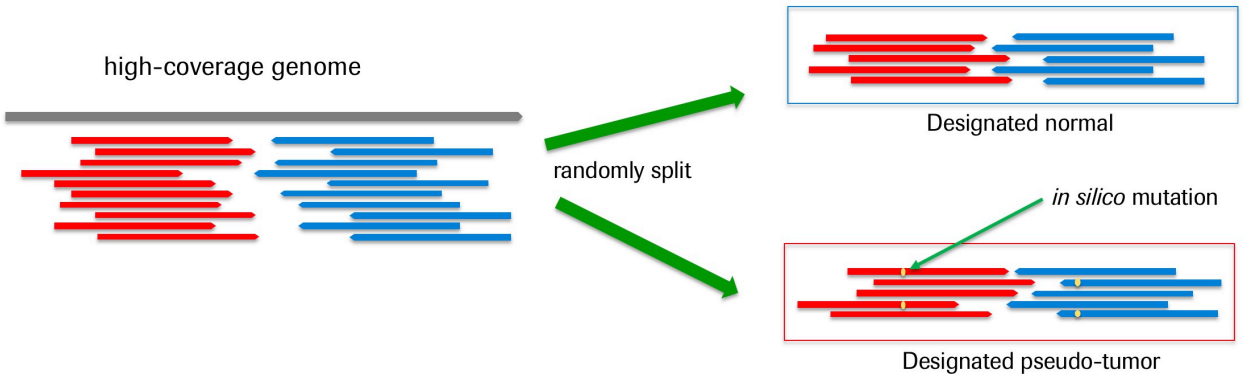
DREAM Somatic Mutation Calling Challenge was an international competition to find algorithms that gave the most accurate performances.

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

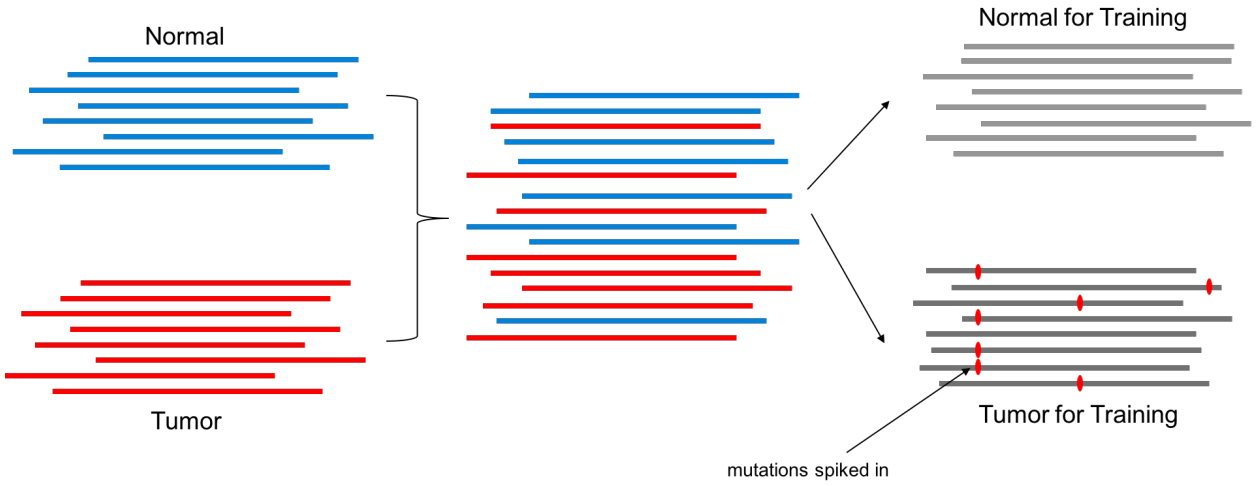


6.2.3 Merge and then split the input tumor and normal BAM files

```
$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.

```
makeSomaticScripts.py paired \
--normal-bam            /ABSOLUTE/PATH/TO/trainingSet/syntheticNormal.bam \
--tumor-bam             /ABSOLUTE/PATH/TO/trainingSet/syntheticTumor.bam \
--genome-reference      /ABSOLUTE/PATH/TO/GRCh38.fa \
--output-directory      /ABSOLUTE/PATH/TO/trainingSet/somaticMutationPipeline \
--dbsnp-vcf             /ABSOLUTE/PATH/TO/dbSNP.GRCh38.vcf \
--truth-snv             /ABSOLUTE/PATH/TO/trainingSet/synthetic_snvs.vcf \
--truth-indel           /ABSOLUTE/PATH/TO/trainingSet/synthetic_indels.leftAlign.vcf \
--threads               16 \
--somaticseq-algorithm xgboost \
--train-somaticseq      \
--run-mutect2 --run-vardict --run-muse --run-lofreq --run-strelka2 --run-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 redundant features
- 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 caller decision.

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*

29. Version 3.0.1

- Fixed a bug that didn't handle Strelka/LoFreq indel calls correctly in *somaticseq/combine_callers.py* module.

30. Version 3.1.0

    - When splitting MuTect2 files into SNV and INDEL, make sure either the ref base or the alt base (but not both) consists of a single base, i.e., discarding stuff like GCAA>GCT.
    - Fixed a bug introduced in v3.0.1 that caused the program to handle .vcf.gz files incorrectly.
    - Incorporated Platypus into paired mode.

31. Version 3.1.1

    - Fixed some bash scripts involved with single-sample multi-thread callers.
    - *vcfModifier/splitVcf.py* to handle multi-allelic calls better for indels.

32. Version 3.2.0

    - Re-wrote in Python some somatic caller run script generators that were once written in bash, at *utilities/dockered_pipelines/makeSomaticScripts.py*. See Section 5 for details.
    - Fixed *setup.py*, even though running *./setup.py install* is optional. You can still run scripts from where you downloaded SomaticSeq.

33. Version 3.2.1

    - Fixed the TA2CG feature in *ada_model_builder_ntChange.R*.

34. Version 3.3.0

    - Added support for xgboost (extreme gradient boosting) as an optional substitute for ada (adaptive boosting).
    - Also fixed the TA2CG feature in *ada_model_predictor.R*
    - Changed tree depth to 16 in ada model training because internal benchmarking found this to be optimal.
    - Modified *utilities/dockered_pipelines/create_tumor_normal_run_scripts.py* and *utilities/dockered_pipelines/create_tumor_only_run_scripts.py* to pre-process bed file for VarDict (i.e., limit each line of bed file to 5000 bp) no matter what. Originally the process is triggered if average bp per line was greater than 50,000 bp. That assumption broke down in multi-threaded tasks, when the final sub bed file contained many decoy or non-human contigs that drove the average bp/line in a bed file below 50,000 bp, and the resulting job caused VarDict to run out of memory. This is to rectify that issue.
    - Occasionally in bamSurgeon workflow, the synthetic_indels.vcf file will contain non-GCTAN characters in the REF column, causing subsequent GATK LeftAlign to fail. Created the *utilities/dockered_pipelines/bamSimulator/bamSurgeon/convert_nonStandardBasesInVcfs.py* script to take care of that.

35. Version 3.4.0

    - Added linguistic sequence complexity (LC) described by Troyanskaya OG *et. al.* as an additional genomic feature [14]. Thus, models trained on this version are not compatible with tsv files created previously becuase it will include a feature not found in previous versions. However, models created from previous version can still be used here. LC is calculated over a 80-bp window spanning the variant site. In addition, LC is also calculated for 80-bp windows on the left and on the right of the variant position, and we record the lower number. The value is converted to Phred in the output.
    - Fixed a bug in xgboost mode where training and testing used different feature sets.

- Change ada classifier's file name from *.ntChange.Classifier.RData to *.ada.Classifier.RData to better distinguish them from xgboost.

36. Version 3.4.1

    - Fixed a bug where indels within 3 bps of a position double-counted indels within 1 bp of the position.

37. Version 3.4.2

    - Modified the linguistic sequence complexity calculation to limit the substring to 20-bp. It increases runtime with no sacrifice of accuracy.
    - Fixed a bug where the indels nearest to a position was not calculated correctly when there are soft-clipped bases in a read.

38. Version 3.5.0

    - Replaced z-scores from scipy's ranksums with p-values from scipy's mannwhitneyu, mostly because the mannwhitneyu corrects for discrete values. Thus, models built prior to this version is no longer compatible with it due to different features.

39. Version 3.5.1

    - Fixed a minor bug when num_caller in *somaticseq/somatic_vcf2tsv.py* and *somaticseq/single_sample_vcf2tsv.py* was not reset when there are multiple variant calls in the same genomic position. So, some variant calls that should not be output into the .tsv because num_caller did not meet the threshold will be output into the .tsv file. However, the features are still reported correctly, so the classifications will still be correct.

40. Version 3.5.2

    - Got around VarDict's latest output VCF file that are incompatible with bedtools by removing the offending lines. Extra steps (may remove later if it becomes unnecessary) were added to somaticseq/combine_callers.py.
    - Noticed that the xgboost script in R is not compatible with version 1.0+ xgboost library in R.

41. Version 3.6.0

    - Re-wrote the XGBoost routine to use the xgboost library in python (somaticseq/somatic_xgboost.py), and made it the default algorithm for SomaticSeq because xgboost in python is orders of magnitudes faster than AdaBoost in R. Added dependencies for python's pandas and xgboost libraries. Thus, if you do not intend to use the AdaBoost package in R (it does not support multi-threaded learning and requires large amount of memory), R is not required. To keep using AdaBoost, make sure to invoke *-algo ada*. As a script, *somatic_xgboost.py* can also be run on its own, with the ability to take in multiple SomaticSeq TSV files. It will combine the TSV files before training (for prediction mode there can only be one input TSV file). Some parametes are also tunable. Run *somatic_xgboost.py train -h* or *somatic_xgboost.py predict -h* to see full options.
    - Remove obsolete *SomaticSeq.Wrapper.sh* and *ssSomaticSeq.Wrapper.sh* scripts. They were obsolete since v3.0.0.

42. Version 3.6.1

    - Re-wrote the *makeSomaticScripts.py* module. It now allows the scripts to be executed in parallel where it was created by invoking *--run-workflow*.
    - Added *makeAlignmentScripts.py* module for alignment workflow.

43. Version 3.6.2

    - Fixed a bug where linguistic sequence complexity could not be calculated for variant positions at the edge of a chromosome.
    - Added *--by-caller* option for *makeSomaticScripts.py*, such that time-consuming tools will be executed first to optimize run time. Be careful about memory usages.

44. Version 3.6.3

    - Change xgboost's default *max_depth* to 8, because after going through some parameters 8 is better than 12 most of the times, though xgboost parameters are all tunable. See options by running in command line: (*somatic_xgboost.py train -h*).
    - Fixed the *–somaticseq-algorithm* option in *makeSomaticScripts.py*. The default is *xgboost* but you can use *ada*.
    - Used *shell=True* option in many *subprocess.call* instances, to allow more complicated arguments to be passed into, e.g., *–action 'qsub -l walltime=100:00:00'*.
    - Moved utilities, genomicFileHandler, and vcfModifier directories into somaticseq/somaticseq to prevent potential package conflicts.

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.

References

[1] Kristian Cibulskis, Michael S Lawrence, Scott L Carter, Andrey Sivachenko, David Jaffe, Carrie Sougnez, Stacey Gabriel, Matthew Meyerson, Eric S Lander, and Gad Getz. Sensitive detection of somatic point mutations in impure and heterogeneous cancer samples. *Nat Biotechnol*, 31(3):213–219, Mar 2013.

[2] Daniel C Koboldt, Qunyuan Zhang, David E Larson, Dong Shen, Michael D McLellan, Ling Lin, Christopher A Miller, Elaine R Mardis, Li Ding, and Richard K Wilson. VarScan 2: somatic mutation and copy number alteration discovery in cancer by exome sequencing. *Genome Res.*, 22(3):568–76, March 2012.

[3] Andrew Roth, Jiarui Ding, Ryan Morin, Anamaria Crisan, Gavin Ha, Ryan Giuliany, Ali Bashashati, Martin Hirst, Gulisa Turashvili, Arusha Oloumi, Marco A Marra, Samuel Aparicio, and Sohrab P Shah. JointSNVMix: a probabilistic model for accurate detection of somatic mutations in normal/-tumour paired next-generation sequencing data. *Bioinformatics*, 28(7):907–13, April 2012.

[4] David E Larson, Christopher C Harris, Ken Chen, Daniel C Koboldt, Travis E Abbott, David J Dooling, Timothy J Ley, Elaine R Mardis, Richard K Wilson, and Li Ding. SomaticSniper: identification of somatic point mutations in whole genome sequencing data. *Bioinformatics*, 28(3):311–7, February 2012.

[5] Zhongwu Lai, Aleksandra Markovets, Miika Ahdesmaki, Brad Chapman, Oliver Hofmann, Robert McEwen, Justin Johnson, Brian Dougherty, J Carl Barrett, and Jonathan R Dry. VarDict: a novel and versatile variant caller for next-generation sequencing in cancer research. *Nucleic Acids Res.*, 44(11):e108, 06 2016.

[6] Yu Fan, Liu Xi, Daniel S T Hughes, Jianjun Zhang, Jianhua Zhang, P Andrew Futreal, David A Wheeler, and Wenyi Wang. MuSE: accounting for tumor heterogeneity using a sample-specific error model improves sensitivity and specificity in mutation calling from sequencing data. *Genome Biol.*, 17(1):178, 08 2016.

[7] Andreas Wilm, Pauline Poh Kim Aw, Denis Bertrand, Grace Hui Ting Yeo, Swee Hoe Ong, Chang Hua Wong, Chiea Chuen Khor, Rosemary Petric, Martin Lloyd Hibberd, and Niranjan Nagarajan. LoFreq: a sequence-quality aware, ultra-sensitive variant caller for uncovering cell-population heterogeneity from high-throughput sequencing datasets. *Nucleic Acids Res.*, 40(22):11189–201, December 2012.

[8] Giuseppe Narzisi, Jason A O'Rawe, Ivan Iossifov, Han Fang, Yoon-Ha Lee, Zihua Wang, Yiyang Wu, Gholson J Lyon, Michael Wigler, and Michael C Schatz. Accurate de novo and transmitted indel detection in exome-capture data using microassembly. *Nat. Methods*, 11(10):1033–6, October 2014.

[9] Sangtae Kim, Konrad Scheffler, Aaron L Halpern, Mitchell A Bekritsky, Eunho Noh, Morten Källberg, Xiaoyu Chen, Yeonbin Kim, Doruk Beyter, Peter Krusche, and Christopher T Saunders. Strelka2: fast and accurate calling of germline and somatic variants. *Nat. Methods*, 15(8):591–594, August 2018.

[10] Donald Freed, Renke Pan, and Rafael Aldana. Tnscope: Accurate detection of somatic mutations with haplotype-based variant candidate detection and machine learning filtering. *bioRxiv*, 2018.

[11] Helga Thorvaldsdottir, James T. Robinson, and Jill P. Mesirov. Integrative Genomics Viewer (IGV): high-performance genomics data visualization and exploration. *Brief Bioinform*, 14(2):178–192, Mar 2013.

[12] Li Tai Fang, Pegah Tootoonchi Afshar, Aparna Chhibber, Marghoob Mohiyuddin, Yu Fan, John C. Mu, Greg Gibeling, Sharon Barr, Narges Bani Asadi, Mark B. Gerstein, and et al. An ensemble approach to accurately detect somatic mutations using somaticseq. *Genome Biology*, 16(1), Sep 2015.

[13] Aaron R. Quinlan and Ira M. Hall. BEDTools: a flexible suite of utilities for comparing genomic features. *Bioinformatics*, 26(6):841–842, 01 2010.

[14] Olga G Troyanskaya, Ora Arbell, Yair Koren, Gad M Landau, and Alexander Bolshoy. Sequence complexity profiles of prokaryotic genomic sequences: A fast algorithm for calculating linguistic complexity. *Bioinformatics*, 18(5):679–688, 2002.