SomaticSeq Documentation

August 29, 2015

1 Introduction

SomaticSeq is a flexible workflow that uses multiple somatic mutation callers to obtain a combined call set, and then use machine learning to distinguish true mutations from false positives from the call set. The manuscript is in preparation. The project page is *http://bioinform.github.io/somaticseq/* .

SomaticSeq.Wrapper.sh is a bash script that calls a series of scripts to combine the output of the somatic mutation caller(s), after the somatic mutation callers are run. Then, depending on what files are fed to SomaticSeq.Wrapper.sh, it will either train the call set into a classifier, predict high-confidence somatic mutations from the call set, or do nothing.

1.1 Dependencies

- All the .py scripts are written in Python 3 (Yes, Python 3. I deserve brownie points from Python developers). The regex package in Python 3 is also required.
- R, plus the ada package in R.
- GATK, SAMtools, SnpEff/SnpSift, and BEDTools
- dbSNP in VCF format. COSMIC may be needed in the future.
- At least one of MuTect/Indelocator, VarScan2, JointSNVMix, SomaticSniper, VarDict, and MuSE. 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.

2 To use SomaticSeq.Wrapper.sh

The SomaticSeq.Wrapper.sh is a wrapper script that calls a series of programs and procedures. It can be run easily if all the dependencies are met. Some parameters are hard-coded but easily edited. In the next section, we described the workflow in more detail, so you are not dependent on this wrapper script. You can either modify this wrapper script or create your own workflow.

2.1 To train data set into a classifier

To create a trained classifier, ground truth files are required for the data sets. There is also an option to include a list of regions to ignore, where the ground truth is not known in those regions.

```
# -M/-I/-V/-v/-J/-S/-D/-U are output VCF files from individual callers.
# -i is also optional.
SomaticSeq.Wrapper.sh -M MuTect/variants.snp.vcf -I Indelocator/variants.indel.vcf
    -V VarScan2/variants.snp.vcf -v VarScan2/variants.indel.vcf -J JointSNVMix2/
    variants.snp.vcf -S SomaticSniper/variants.snp.vcf -D VarDict/variants.vcf -U
    MuSE/variants.snp.vcf -N matched_normal.bam -T tumor.bam -R ada_model_builder.R
     -g human_b37.fasta -c cosmic.b37.v71.vcf -d dbSNP.b37.v141.vcf -s $PATH/TO/DIR
    /snpSift -G $PATH/TO/GenomeAnalysisTK.jar -i ignore.bed -Z truth.snp.vcf -z
    truth.indel.vcf -o $OUTPUT_DIR
```

SomaticSeq.Wrapper.sh supports any combination of the somatic mutation callers we have incorporated into the workflow, so -M/-I/-V/-v/-J/-S/-D/-U are all optional parameters. SomaticSeq will run based on the output VCFs you have provided. It will train SNV and/or INDEL if you provide the truth.snp.vcf and/or truth.indel.vcf file(s).

2.2 To predict somatic mutation based on trained classifiers

```
# The *RData files are trained classifier from the training mode.
SomaticSeq.Wrapper.sh -M MuTect/variants.snp.vcf -I Indelocator/variants.indel.vcf
    -V VarScan2/variants.snp.vcf -v VarScan2/variants.indel.vcf -J JointSNVMix2/
    variants.snp.vcf -S SomaticSniper/variants.snp.vcf -D VarDict/variants.vcf -U
    MuSE/variants.snp.vcf -N matched_normal.bam -T tumor.bam -R ada_model_predictor
    .R -C sSNV.Classifier.RData -x sINDEL.Classifier.RData -g human_b37.fasta -c
    cosmic.b37.v71.vcf -d dbSNP.b37.v141.vcf -s $PATH/TO/DIR/snpSift -G $PATH/TO/
    GenomeAnalysisTK.jar -o $OUTPUT_DIR
```

3 The step-by-step SomaticSeq Workflow

We'll describe the workflow here, so you may modify the workflow and/or create your own workflow instead of using the wrapper script described previously.

3.1 Combine the call sets

We use GATK CombineVariants to combine the VCF files from different callers. To make them compatible with GATK, the VCFs must be modified. A somatic call is also tagged with the tool names, so the combined VCF retains those information.

1. Modify MuTect and/or Indelocator output VCF files. Somatic calls will be attached the tag 'CGA' in the INFO. Since MuTect's output VCF do not always put the tumor and normal samples in the same columns, the script uses samtools extract sample name information from the BAM files, and then determine which column belongs to the normal, and which column belongs to the tumor.

```
# Modify MuTect's output VCF
# -type snp for MuTect, and -type indel for Indelocator.
modify_MuTect.py -type snp -infile input.vcf -outfile output.vcf -nbam normal.
    bam -tbam tumor.bam

# If samtools is not in the PATH:
modify_MuTect.py -type snp -infile input.vcf -outfile output.vcf -nbam normal.
    bam -tbam tumor.bam -samtools $PATH/TO/samtools
```

Alternatively, you can supply the normal and tumor sample names, instead of supplying the BAM files:

```
# Modify MuTect's output VCF
# -type snp for MuTect, and -type indel for Indelocator.
modify_MuTect.py -type snp -infile input.vcf -outfile output.vcf -nsm
    NormalSampleName -tsm TumorSampleName
```

2. Modify VarScan's output VCF files to be rigorously concordant to VCF format standard, and to attach the tag 'VarScan2' to somatic calls.

```
# Do it for both the SNV and INDEL
modify_VJSD.py -method VarScan2 -infile input.vcf -outfile output.vcf
```

3. JointSNVMix2 does not output VCF files. In our own workflow, we have already converted its text file into a basic VCF file with an 2 awk one-liner, which you may see in the Run_5_callers directory, which are:

```
# To avoid text files in 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
```

After that, you'll also want to sort the VCF file. Now, to modify that basic VCF into something that will be compatible with other VCF files under GATK CombineVariants:

```
modify_VJSD.py -method JointSNVMix2 -infile input.vcf -outfile output.vcf
```

4. Modify SomaticSniper's output:

```
modify_VJSD.py -method SomaticSniper -infile input.vcf -outfile output.vcf
```

5. VarDict has both SNV and INDEL, plus some other variants in the same VCF file. Our script will create two files, one for SNV and one for INDEL, while everything else is ignored for now. By default, LikelySomatic/StrongSomatic and PASS calls will be labeled VarDict. However, in our SomaticSeq paper, based on our experience in DREAM Challenge, we implemented two custom filters to relax the VarDict tagging criteria.

```
# Default VarDict tagging criteria, only PASS (and Likely or Strong Somatic):
modify_VJSD.py -method VarDict -infile intput.vcf -outfile output.vcf

# When running VarDict, if var2vcf_paired.pl is used to generate the VCF file,
     you may relax the tagging criteria with -filter paired
modify_VJSD.py -method VarDict -infile intput.vcf -outfile output.vcf -filter
    paired

# When running VarDict, if var2vcf_somatic.pl is used to generate the VCF file
    , you may relax the tagging criteria with -filter somatic
modify_VJSD.py -method VarDict -infile intput.vcf -outfile output.vcf -filter
    somatic
```

   In the SomaticSeq paper, -filter somatic was used because var2vcf_somatic.pl was used to generate VarDict's VCF files. In the SomaticSeq.Wrapper.sh script, however, -filter paired is used because VarDict authors have since recommended var2vcf_paired.pl script to create the VCF files. While there are some differences (different stringencies in some filters) in what VarDict labels as PASS between the somatic.pl and paired.pl scripts, the difference is miniscule after applying our custom filter (which relaxes the filter, resulting in a difference about 5 calls out of 15,000).

   The output files will be snp.output.vcf and indel.output.vcf.

6. MuSE was not a part of our analysis in the SomaticSeq paper. We have implemented it later.

```
modify_VJSD.py -method MuSE -infile input.vcf -outfile output.vcf
```

7. Finally, with the VCF files modified, you may combine them with GATK CombineVariants: one for SNV and one for INDEL separately. There is no particular reason to use GATK CombineVariants. Other combiners should also work. The only useful thing here is to combine the calls, and preserve the tags we have written into each individual VCF file's INFO.

```
# Combine the VCF files for SNV. Any or all of the VCF files may be present.
# -nt 12 means to use 12 threads in parallel
java -jar $PATH/TO/GenomeAnalysisTK.jar -T CombineVariants -R genome.GRCh37.fa
     -nt 12 --setKey null --genotypemergeoption UNSORTED -V mutect.vcf -V
    varscan.snp.vcf -V jointsnvmix.vcf -V snp.vardict.vcf -V muse.vcf --out
    CombineVariants.snp.vcf
java -jar $PATH/TO/GenomeAnalysisTK.jar -T CombineVariants -R genome.GRCh37.fa
     -nt 12 --setKey null --genotypemergeoption UNSORTED -V indelocator.vcf -V
     varscan.snp.vcf -V indel.vardict.vcf --out CombineVariants.indel.vcf
```

8. Use SnpSift to add dbSNP information to the VCF file, since dbSNP information is part of training feature set.

```
java -jar $PATH/TO/SnpSift.jar annotate dbSNP141.vcf CombineVariants.snp.vcf >
     dbSNP.CombineVariants.snp.vcf
java -jar $PATH/TO/SnpSift.jar annotate dbSNP141.vcf CombineVariants.indel.vcf
     > dbSNP.CombineVariants.indel.vcf
```

Right now, we do not use COSMIC or functional annotation as a part of training feature, but we do have them in the workflow for "future-proofing." We may decide to use those features in the future when we have better data sets for training.

```
java -jar $PATH/TO/SnpSift.jar annotate cosmic71.vcf dbSNP.CombineVariants.vcf
      > COSMIC.dbSNP.CombineVariants.vcf
java -jar $PATH/TO/SnpSift.jar GRCh37.75 COSMIC.dbSNP.CombineVariants.vcf >
    EFF.COSMIC.dbSNP.CombineVariants.vcf
```

9. This procedure annotates the caller consensus by putting the tool names into the SOURCES in the INFO. You should also use -mincaller 1 to only keep calls where at least one caller has called it somatic. Vast majority of the calls in the previous merged VCF files were REJECT or GERMLINE calls, and may run out of memory in model training if all are included. It also does some rudimentary variant scoring, but the scoring is not utilized in SomaticSeq. You need to run it once for SNV and once for INDEL.

```
# Use -tools to indicate what call sets were combined. CGA=MuTect/Indelocator
# In this one, 6 tools were used for SNV
score_Somatic.Variants.py -tools CGA VarScan2 JointSNVMix2 SomaticSniper
    VarDict MuSE -infile EFF.COSMIC.dbSNP.CombineVariants.snp.vcf -mincaller 1
     -outfile BINA_somatic.snp.vcf

# 3 tools for INDEL
score_Somatic.Variants.py -tools CGA VarScan2 VarDict -infile EFF.COSMIC.dbSNP
    .CombineVariants.indel.vcf -mincaller 1 -outfile BINA_somatic.indel.vcf
```

And now, the call sets are combined into one VCF file for SNV and one VCF file for INDEL.

3.2 For model training: process and annotate the VCF files (union of call sets)

This step makes sense for model training. The workflow in SomaticSeq.Wrapper.sh allows for an exclusion region, e.g., we don't care for anything inside the exclusion region. DREAM Challenge had exclusion regions, e.g., blacklisted regions, etc. Alternatively, you can use an inclusion region instead, to obtain a list of variants that you have done experimental validation, so you know which ones are true mutations and which ones are false positives, to facilitate model training. It will then annotate the optionally processed VCF file (a subset of the original VCF file, which is optional) against a ground truth VCF file where only true mutations are included. Important: any variant inside the merged VCF file (processed or otherwise) but not inside the ground truth VCF file will be annotated as a false positive. Any variant that appears in both VCF file will be annotated as a true positive. Anything that appears in the ground truth but not in the merged VCF file will be annotate as a true negative. The output file for annotation is a VCF, but may not be properly formatted, which is okay us.

```
# In the DREAM_Stage_3 directory, we have included an exclusion region BED file as
    an example
# This command uses BEDtools to rid of all calls in the exclusion region
intersectBed -header -a BINA_somatic.snp.vcf -b ignore -v > somatic.snp.processed.
    vcf
intersectBed -header -a BINA_somatic.indel.vcf -b ignore -v > somatic.indel.
    processed.vcf

# This script will annotate the processed VCF files against the ground truth.
# DREAM Challenge Stage 3 ground truth VCF files are included in the DREAM_Stage_3
    directory.
tally_MyVCF_vs_Truth.py -truth truth.snp.vcf -myvcf somatic.snp.processed.vcf -
    outfile annotated.snp.vcf
tally_MyVCF_vs_Truth.py -truth truth.indel.vcf -myvcf somatic.indel.processed.vcf -
    outfile annotated.indel.vcf
```

3.3 Convert the VCF file, annotated or otherwise, into a tab separated file

This script works for all VCF files. If the input VCF file is annotated with ground truth, the output TSV file will have 1 and 0 for true mutations and false positives, and can be used for model training (as well as mutation prediction, so you can evaluate the prediction accuracy). Otherwise, those values will be "nan" and cannot be used for training (but can still be used for mutation prediction).

The script extracts information from VCF files by SAMTools, HaplotypeCallers, and/or individual VCF files created by the individual callers.

```
# -sniper / -varscan / -jsm / -vardict / -muse are optional. We do not extract any
    additional information from MuTect's output VCF file, so -mutect is not used.
SSeq_merged.vcf2tsv.py -fai genome.GRCh37.fa.fai -myvcf somatic.snp.processed.vcf -
    varscan VarScan2/variants.snp.vcf -jsm JSM2/variants.vcf -sniper SomaticSniper/
    variants.vcf -vardict VarDict/snp.variants.vcf -samT samT.vcf -samN samN.vcf -
    haploT haploT.vcf -haploN haploN.vcf -outfile Ensemble.sSNV.tsv
```

To speed things up a little bit, SomaticSeq.Wrapper.sh uses FIFO for the VCF files generated by SAMtools and HaplotypeCaller:

```
# Make fifo for SAMtools and HaplotypeCaller outputs:
mkfifo samN.vcf.fifo samT.vcf.fifo haploN.vcf.fifo haploT.vcf.fifo

# For SNV, extract information only in the somatic.snp.processed.vcf, and exclude
    INDEL calls by SAMtools.
samtools mpileup -B -uf genome.GRCh37.fa normal.bam -l somatic.snp.processed.vcf |
    bcftools view -cg - | egrep -wv 'INDEL' > samN.vcf.fifo &
samtools mpileup -B -uf genome.GRCh37.fa tumor.bam -l somatic.snp.processed.vcf |
    bcftools view -cg - | egrep -wv 'INDEL' > samT.vcf.fifo &

# Same concept for HaplotypeCaller:
java -Xms8g -Xmx8g -jar $PATH/TO/GenomeAnalysisTK.jar -T HaplotypeCaller --dbsnp
    dbSNP.b37.v141.vcf --reference_sequence genome.GRCh37.fa -L somatic.snp.
    processed.vcf --emitRefConfidence BP_RESOLUTION -I normal.bam --out /dev/stdout
     | awk -F "\t" '$0 ~ /^#/ || ( $4 ~ /^[GCTA]$/ && $5 !~ /[GCTA][GCTA]/ )' >
    haploN.vcf.fifo &
java -Xms8g -Xmx8g -jar $PATH/TO/GenomeAnalysisTK.jar -T HaplotypeCaller --dbsnp
    dbSNP.b37.v141.vcf --reference_sequence genome.GRCh37.fa -L somatic.snp.
    processed.vcf --emitRefConfidence BP_RESOLUTION -I tumor.bam --out /dev/stdout
    | awk -F "\t" '$0 ~ /^#/ || ( $4 ~ /^[GCTA]$/ && $5 !~ /[GCTA][GCTA]/ )' >
    haploT.vcf.fifo &

# Stream those .fifo into my script
SSeq_merged.vcf2tsv.py -fai genome.GRCh37.fa.fai -myvcf somatic.snp.processed.vcf -
    varscan VarScan2/variants.snp.vcf -jsm JSM2/variants.vcf -sniper SomaticSniper/
    variants.vcf -vardict VarDict/snp.variants.vcf -samT samT.vcf.fifo -samN samN.
    vcf.fifo -haploT haploT.vcf.fifo -haploN haploN.vcf.fifo -outfile Ensemble.sSNV
    .tsv
```

That was for SNV, INDEL is almost the same thing. After version 1.1, DP4 (i.e., 4 numbers related to read depth: forward reference read counts, reverse reference read counts, forward alternate read counts, and reverse alternate read counts) for INDEL are extracted from SAMtools mpileup instead of SAMtools VCF. SAMtools VCF has these information for INDELs only if it calls an INDEL variant at the site, so it's more appropriate to grab that information directly from pileup. In addition, if depth is not found in SAMtools or HaplotypeCaller (simply because they do not call INDEL variants in those sites), that's directly extracted from pileup as well.

If pileup files are supplied for SNV, they will take precedence over SAMtools and HaplotypeCaller VCF, but for SNV there is no advantage running extra processes because SAMtools gets all the DP4 stats for SNV.

```
# Make fifo for SAMtools and HaplotypeCaller outputs:
mkfifo samN.indel.vcf.fifo samT.indel.vcf.fifo haploN.indel.vcf.fifo haploT.indel.
    vcf.fifo

# For INDEL, extract information only in the somatic.indel.processed.vcf, and only
    use INDEL calls.
samtools mpileup -B -uf genome.GRCh37.fa normal.bam -l somatic.indel.processed.vcf
    | bcftools view -cg - | egrep '^#|INDEL' > samN.indel.vcf.fifo &
samtools mpileup -B -uf genome.GRCh37.fa tumor.bam -l somatic.indel.processed.vcf |
     bcftools view -cg - | egrep '^#|INDEL' > samT.indel.vcf.fifo &

# DP4 comes from pileup files for INDELS (-Q 13 by default):
samtools mpileup -B -q 1 -f genome.GRCh37.fa normal.bam -l somatic.indel.processed.
    vcf > plN.indel.pileup.fifo &
samtools mpileup -B -q 1 -f genome.GRCh37.fa tumor.bam -l somatic.indel.processed.
    vcf > plT.indel.pileup.fifo &

# Same concept for HaplotypeCaller. Keep only INDELs.
```

```
java -Xms8g -Xmx8g -jar $PATH/TO/GenomeAnalysisTK.jar -T HaplotypeCaller --dbsnp
    dbSNP.b37.v141.vcf --reference_sequence genome.GRCh37.fa -L somatic.indel.
    processed.vcf --emitRefConfidence BP_RESOLUTION -I normal.bam --out /dev/stdout
     | awk -F "\t" '$0 ~ /^#/ || $4 ~ /[GCTA][GCTA]/ || $5 ~ /[GCTA][GCTA]/' >
    haploN.indel.vcf.fifo &
java -Xms8g -Xmx8g -jar $PATH/TO/GenomeAnalysisTK.jar -T HaplotypeCaller --dbsnp
    dbSNP.b37.v141.vcf --reference_sequence genome.GRCh37.fa -L somatic.indel.
    processed.vcf --emitRefConfidence BP_RESOLUTION -I tumor.bam --out /dev/stdout
    | awk -F "\t" '$0 ~ /^#/ || $4 ~ /[GCTA][GCTA]/ || $5 ~ /[GCTA][GCTA]/' >
    haploT.indel.vcf.fifo &

# Stream those .fifo into my script
SSeq_merged.vcf2tsv.py -fai genome.GRCh37.fa.fai -myvcf somatic.indel.processed.vcf
     -varscan VarScan2/variants.snp.vcf -vardict VarDict/indel.variants.vcf -samT
    samT.indel.vcf.fifo -samN samN.indel.vcf.fifo -plT plT.indel.pileup.fifo -plN
    plN.indel.pileup.fifo -haploT haploT.indel.vcf.fifo -haploN haploN.indel.vcf.
    fifo -outfile Ensemble.sINDEL.tsv
```

At the end of this, Ensemble.sSNV.tsv and Ensemble.sINDEL.tsv are created.

3.4 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 --no-save --args Ensemble.sSNV.tsv < ada_model_builder.R
R --no-save --args Ensemble.sINDEL.tsv < ada_model_builder.R
```

Ensemble.sSNV.tsv.Classifier.RData and Ensemble.sINDEL.tsv.Classifier.RData will be created from model training.

Mutation prediction:

```
# Mutation prediction:
R --no-save --args Ensemble.sSNV.tsv.Classifier.RData Ensemble.sSNV.tsv Trained.
    sSNV.tsv < ada_model_predictor.R
R --no-save --args Ensemble.sINDEL.tsv.Classifier.RData Ensemble.sINDEL.tsv Trained
    .sINDEL.tsv < ada_model_predictor.R
```

After mutation prediction, if you feel like it, you may convert Trained.sSNV.tsv and Trained.sINDEL.tsv into VCF files.

```
# If you want probability above 0.7 labeled PASS (-pass 0.7), and between 0.1 and
    0.7 labeled LowQual (-low 0.1):
# To include calls below 0.1 as REJECTS (-all)# To convert probability values (
    between 0 to 1) into Phred scale in the QUAL column in the VCF file (-phred)
SSeq_tsv2vcf.py -tsv Trained.sSNV.tsv -vcf Trained.sSNV.vcf -pass 0.7 -low 0.1 -all
     -phred
SSeq_tsv2vcf.py -tsv Trained.sINDEL.tsv -vcf Trained.sINDEL.vcf -pass 0.7 -low 0.1
    -all -phred
```

4 Release Notes

4.1 version 1.0

Version used to generate data in the manuscript

4.2 version 1.1

Improved the SomaticSeq.Wrapper.sh script to automate either training or prediction procedure.

4.3 Current

Have implemented the following improvement:

- SSeq_merged.vcf2tsv.py now accepts pileup files, so it can extract read depth and DP4 (reference forward, reference reverse, alternate forward, and alternate reverse) information (mainly for INDELs), since the SAMtools or HaplotypeCaller generated VCFs hardly contain any INDEL information. 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 for INDELs).
- INDEL length now positive for insertions and negative for deletions.

5 Contact Us

For suggestions and bug reports, please email li_tai.fang@bina.roche.com.