# AVA,Dx pipeline

## Input and output

### Input:

- VCF file (GRCh37/hg19)
  - e.g. `source.vcf.gz`
- cross-validation data split schemes with class labels
  - e.g. `cv-scheme.txt` with columns '*SampleID*', '*Phenotype*', '*fold*'

### Output:

- a *gene score* table
  - e.g. `GeneScoreTable_normed.txt` and/or `GeneScoreTable_unnormed.txt`
    - two schemes available to choose: *sum* or *product* (see details below)
    - `_normed` means normalizing every *gene score* by protein length
- selected genes
  - e.g. `10F-CV-KS-selectedGenes.csv`
    - `10F-CV` means 10-fold cross-validation
    - `-KS-` means Kolmogorov–Smirnov test was used to select genes
- model performance
  - e.g. `10F-CV-rf-performance.xlsx`
    - `10F-CV` means 10-fold cross-validation
    - `-rf-` means random forest was used to build models

## Prerequisite

- R and packages (data.table, tydiverse, seqinr, stringr, EthSEQ, SNPRelate, e1071, caret)
- python2
- tabix
- bcftools
- ANNOVAR
- SNAP or snapfun.db
- Slurm cluster system (e.g. Amarel) for submitting jobs parallelly

## About

- The current workflow works primarily with hg19. For hg18, I recommend lifting over to hg19 first. For hg38, I recommend updating all reference databases used in this pipeline.
- This pipeline is currently for regular VCF file input (modifications needed for gVCF files).
- Manual interpretation of quality outliers, ethnicity, is needed and recommended.
- When input VCF contains variants with no SNAP score records in the `snapfun.db`, the snapfun.db needs to be updated. SNAP scores can also be calculated by running SNAP.

## Step 1: Variant QC

Analyses in AVA,Dx and many other methods need stringent QC of the VCF file. Because all genes are taken into consideration in FS and model building, and artifacts in the data could lead to biased results. This very first step removes "bad" variants in the

VCF file by removing variant loci.

> Note that thresholds used below are all determined empirically and quite stringent, as there is no "gold standard" in this process. I recommend setting these thresholds according to specific needs. Easing the thresholds can help retain more informative variants at the risk of keeping more false positive calls.

- Sometimes the VCF file contains more individuals than needed. In this case, extract individuals of interest (diseased and healthy individuals of interest) in `sampleID.txt`.

```
bcftools view -S sampleID.txt source.vcf.gz -Oz -o source_samp.vcf.gz
```

- Remove sites which did not pass the VQSR standard.

```
bcftools filter -i 'FILTER="PASS"' source_samp.vcf.gz -Oz -o source_samp_pass.vcf.gz
```

- Split SNV and InDel calls to separated files because they use different QC thresholds. Current AVA,Dx works mainly with SNPs. InDels need another set of standards for QC.

```
bcftools view --types snps source_samp_pass.vcf.gz -Oz \
  -o source_samp_pass_snps.vcf.gz
bcftools view --types indels source_samp_pass.vcf.gz -Oz \
  -o source_samp_pass_indels.vcf.gz
```

- Remove sites by site-wise quality. Default good site-wise qualities here are: QUAL > 30, mean DP >= 6, mean DP <= 150.

> QUAL is an internal metric generated by GATK for site quality assessment. Low DP means that this variant was called based on very few evidences, and thus low quality. High DP is not always good either, as it is likely from sequence fragments of repeated regions.

```
bcftools view -i 'QUAL>30 & AVG(FMT/DP)<=150 & AVG(FMT/DP)>=6' source_samp_pass_snps.vcf.gz \
  -Oz -o source_samp_pass_snps_site-v.vcf.gz
```

- Check individual call quality. Default good individual call qualities are: AB > 0.3 and AB < 0.7, GQ > 15, DP > 4; all bad individual calls are converted into missing; then, low call rate is determined as a call rate < 80%, *i.e.* missing rate >= 20%. Variant sites with a high missing rate are removed.

> Allele Balance (AB) is an important quality metric that has been ignored before. Some VCF file does not have the AB field by default. In that case, `filterVCF_by_ABAD.py` takes the AD field and calculates AB.

```
python filterVCF_by_ABAD.py \
  source_samp_pass_snps_site-v.vcf.gz \
  source_samp_pass_snps_site-v_gt-v.vcf.gz
```

- Lastly, gnomAD filter: filtering out variants that do not exist in the gnomAD database. The gnomAD reference used here is the ANNOVAR gnomAD file. Check the input path of the two reference files before running the script.

> Note that files `hg19_gnomad_exome_allAFabove0.txt.gz`, `hg19_gnomad_genome_allAFabove0.txt.gz`, and `tabix` command tool is required to run this script. This step could be slow if there're a lot of variants to check. The two files are not provided here due to their sizes; they can be sent upon request should you decide to run it.

```
# Conver the chromosome annotation if the chromosomes are recorded as "chr1" instead of "1":
bcftools annotate --rename-chrs chr_to_number.txt input.vcf.gz -Oz -o input_rmchr.vcf.gz
# Then remove variants that are not in gnomAD database:
python filterVCF_by_gnomAD.py input_rmchr.vcf.gz output.vcf.gz
```

> gnomAD also contains low quality calls. For example, variant 1-30548-T-G is covered in fewer than 50% of individuals in exomes and genomes (gnomAD v2.1.1) and the allele balance are skewed in some individuals. Specifically, this variant has a "." in the exome reference file ( `hg19_gnomad_exome.txt` ). But it will be kept as long as the genome reference ( `hg19_gnomad_genome.txt` ) has a record of this variant.
>
> To generate `hg19_gnomad_exome_allAFabove0.txt.gz` and `hg19_gnomad_genome_allAFabove0.txt.gz` files, use ANNOVAR's gnomAD database. Note that the current db for gnomAD at ANNOVAR only has hg19 assembly. Example command to download the gnomAD database is: `annotate_variation.pl -downdb gnomad_exome humandb -buildver hg19` and `annotate_variation.pl -downdb gnomad_genome humandb -buildver hg19`.
> After downloading, you'll find `hg19_gnomad_exome.txt` and `hg19_gnomad_genome.txt` in the humandb/ folder. The `hg19_gnomad_exome_allAFabove0.txt.gz` and `hg19_gnomad_genome_allAFabove0.txt.gz` files were generated by extracting the first two columns (chr and position) of `hg19_gnomad_exome.txt` and `hg19_gnomad_genome.txt` files with an allele frequency of ALL (6th columns of `hg19_gnomad_exome.txt` and `hg19_gnomad_genome.txt` ) above zero. Then use `bgzip` and `tabix` to zip and index the output files into `.txt.gz` and `.txt.gz.tbi` format.

## Step 2: Individual QC

Removing individual outliers by checking quality metrics, sample ethnicities, and relatedness.

### Quality check:

- Check quality outliers by examine nRefHom, nNonRefHom, nHets, nTransitions, nTransversions, average depth, nSingletons, and nMissing:

```
# Output quality metrics after variant QC:
bcftools stats -v -s - \
  source_samp_pass_snps_site-v_gt-v.vcf.gz > \
  source_samp_pass_snps_site-v_gt-v.vcf.stats.txt
# Draw individual quality figure:
Rscript stats_quality_pca.R \
  -f source_samp_pass_snps_site-v_gt-v.vcf.stats.txt
```

> Above script output a PCA plot of samples clustered by the first two PCs in terms of their quality metrics *after variant QC*. **User needs to pick up the outliers and decide whether to keep them in later analyses.**

### Ethnicity check:

- If there were clear annotations of sample ethnicity in sampling collection step, this can be skipped. But if ethnicity info is missing, annotate ethnicity with EthSEQ:

```
# convert format for EthSEQ input:
bcftools annotate --remove 'ID,INFO,FORMAT' \
  source_samp_pass_snps_site-v_gt-v.vcf.gz \
  | bcftools view --no-header -Oz \
  -o source_EthSEQinput.vcf.gz
# Run EthSEQ:
Rscript ethnicity_EthSEQ.R \
  source_EthSEQinput.vcf.gz \
  /path/to/output/folder
```

- If the VCF file contains more than 500 samples:

```
# extract sample IDs to chunks:
bcftools query \
  -l source_samp_pass_snps_site-v_gt-v.vcf.gz > sample_list.txt
# split sample to chunks:
csplit sample_list.txt 500  # outputs from xx00 to xx0n
# perform below for EVERY chunk of samples:
bcftools view -S xx00 source_samp_pass_snps_site-v_gt-v.vcf.gz \
  | bcftools annotate --remove 'ID,INFO,FORMAT' \
  | bcftools view --no-header \
  -Oz -o source_xx00_EthSEQinput.vcf.gz
# Run EthSEQ:
# "export R_MAX_VSIZE=32000000000" can be used to increase memory before running below for larger datasets
Rscript ethnicity_EthSEQ.R \
  source_xx00_EthSEQinput.vcf.gz \
  /path/to/output/folder/for/chunk/xx00
```

> Detailed results of ethnicity check are in `/path/to/output/folder/Report.txt`.

- Get sample IDs of European descent (EUR) samples:

```
# get sample_list.txt:
bcftools query \
  -l source_samp_pass_snps_site-v_gt-v.vcf.gz > sample_list.txt
# or use xx00 file if the sample IDs were split into chunks
Rscript ethnicity_EthSEQ_summary.R /path/to/output/folder/Report.txt sample_list.txt /path/to/output/folder
```

> Above step returns two files: `sampleID_closest_EUR.txt` and `sampleID_inside_EUR.txt`. `sampleID_inside_EUR.txt` contains the sample IDs for all EUR individuals in the dataset. `sampleID_closest_EUR.txt` contains sample IDs that are *close to* EUR according to their variants. **User needs to pick the sample IDs of interest for later analyses.**

### Relatedness check:

- Samples collected from a study might accidentally have related pairs. This step checks relatedness within datasets using SNPRelate package.

> A kinship value of ~0.5 indicates first-degree relatedness, e.g. parent-child.

```
Rscript relatedness.R \
  -i source_samp_pass_snps_site-v_gt-v.vcf.gz \
  -g source_samp_pass_snps_site-v_gt-v.gds \
  -c 0.3 \
  -o /path/to/output/folder
```

The output folder contains 3 files: `IBD_histogram.pdf`, `IBD.txt`, and `IBD_related.txt`. The histogram shows the distribution of kinship values of all individual pairs from the input VCF. `IBD.txt` is a complete table of the kinship values. `IBD_related.txt` only contains related pairs per the `-c` cutoff. **One individual from each related pair should be removed for further analysis.**

### Remove individual outliers:

- All outlier sample IDs should be combined from the above PCA, ethnicity annotation, and relatedness check into a file `outliers.txt` (one ID per row). Then remove individual outliers by:

```
bcftools -S ^outliers.txt \
  source_samp_pass_snps_site-v_gt-v.vcf.gz -Oz \
  -o source_samp_pass_snps_site-v_gt-v_ind-clean.vcf.gz
```

## Step 3: Query/Calculate SNAP scores for all variants

- Get all variant annotations with ANNOVAR for cleaned VCF:

```
# Convert VCF file into ANNOVAR input format:
./convert2annovar.pl -format vcf4old source_samp_pass_snps_site-v_gt-v_ind-clean.vcf.gz \
  -outfile source_samp_pass_snps_site-v_gt-v_ind-clean.avinput
# Annotate using hg19 RefSeq:
./annotate_variation.pl -buildver hg19 source_samp_pass_snps_site-v_gt-v_ind-clean.avinput humandb/
```

> `-format vcf4old` is important to get **all** variants in the VCF. As of May 2020, the latest RefSeq reference ANNOVAR uses is *2019Oct24*. All exonic variant will be recorded in the `*.exonic_variant_function` file.

AVA,Dx has a flat file `Mutations.mutOut` to store SNAP scores for all variants from *previous* exome datasets. But for a new dataset, it is very likely that it contains some *new* variants and the `Mutations.mutOut` file does not have their SNAP scores. In this case, perform below step to **either** query SNAP score from `snapfun.db` **or** calculate SNAP score by running SNAP.

### Option 1 - Make `snapfun.db` query file for missing SNAPs (TBF):

Query missing SNAP scores from the snap score database.

- To make query files, extract all variants from `*.exonic_variant_function` to query snap scores from `snapfun.db`. The `db` folder already contains a file (`Mutations.mutOut`) of pre-calculated SNAP scores for variants from previous studies. Below steps will generate query file for variants which are not included in `Mutations.mutOut`.

```
# RE-WRITE SCRIPT NEEDED:
# Make query file:
Rscript exonic_variant_function2snap_query.R \
  -e /path/to/source_samp_pass_snps_site-v_gt-v_ind-clean.avinput.exonic_variant_function \
  -m /path/to/map_RefSeq_and_UniProt.csv \
  -s /path/to/Mutations.mutOut \
  -v id \  # id or sequence
  -o /path/to/out
```

This outputs two files (`query_seq.fa` and `query_mut.in`) into the output folder designated by `-o`.

> ANNOVAR annotates variants to RefSeq mRNA transcripts (e.g. NM_001). The corresponding protein entry (e.g. NP_098) can be retrieved from NCBI database. So for each exonic variant, we know its genomic and genetic location, the mutation (from what to what), and corresponding RefSeq mRNA transcript and protein accesion numbers from the annotation.
>
> Current `snapfun.db` is based on UniProt protein records (e.g. P11310, the ID for entry name ACADM_HUMAN). There are around a quarter of protein sequences do not match between RefSeq protein and UniProt protein. Therefore, the script `exonic_variant_function2snap_query.R` can only output the query file of those protein that can be mapped to UniProt sequences.
>
> Alternatively, to query for SNAP scores for unmappable variants, we can use protein sequences instead of UniProt IDs as query inputs by assigning `-v` to `sequence`.

- After obtaining SNAP scores for "new" variants, update the `Mutations.mutOut` file by:

```
cat /path/to/Mutations.mutOut query_result.txt > /path/to/Mutations.mutOut
```

### Option 2 - Make SNAP input files for missing SNAPs:

The `check_missing_SNAP.R` script generates SNAP input files (mutation files) for missing SNAP.

```
Rscript check_missing_SNAP.R \
   -f /path/to/*.exonic_variant_function \
   -s /path/to/db/Mutations.mutOut \
   -l /path/to/db/Transcript-ProtLength.csv \
   -o /path/to/output/folder
```

Above step generates multiple files with RefSeq transcript as their names (*e.g.* NM_001) and mutations in each file. The fasta files for SNAP input can be retrieved at NCBI Batch Entrez.

To run SNAP on Amarel, prepare two input files for each protein: (1) mutation files; (2) fasta sequence.

```
# e.g. mutation file (NM_001.mut):
A2V
D3M
# e.g. fasta sequence (NM_001.fasta):
>proteinA
MADVAMD
```

> There will still be a small part of variants with no SNAP scores even after above query/calculation.
>
> Possible reasons are:
>
> - The `snapfun.db` does not have pre-calculated scores for this protein sequence or this variant.
> - The variants were lost during mapping from RefSeq to UniProt IDs, *e.g.* the records simply do not match.
> - The variants have wrong annotations, *e.g.* `.exonic_variant_function` file says N639D but the protein sequence has D as the REF aa at position 639.
>
> Whichever the reason is, this should be a very small portion of all variants in the input dataset. Ideally, the exact same reference should be used throughout the analysis starting from variant calling to AVA,Dx. Also, ideally there are no mismatches between database records. However, this is not the case, and there is no perfect way to deal with this issue.

## Step 4: Gene score calculation

- Convert cleaned VCF to **individual** ANNOVAR annotation files by:

```
./convert2annovar.pl -format vcf4 \
  source_s-selected_v-PASS_snps_site-v-Q30-minavgDP6-maxavgDP150_gt-v-DP4-AB37-GQ15-MR20perc_ind-cleaned.vcf.gz \
  -outfile /path/to/output/folder/sample \
  -allsample
```

The `vcf4` and `-allsample` arguments specify that one `sample*.avinput` output file are generated for **every individual** in the VCF, *i.e.* multiple output files.

> *Troubleshooting:* `convert2annovar.pl` script opens new files simultaneously for all samples, and when the sample number exceeds the maximum number (2,048 to be set by `ulimit -n 2048`), the script returns error because it cannot open a larger number of files. If sample number exceeds 2048, split the VCF file into chunks of samples and run `convert2annovar.pl` separately for each chunk.

- Next, annotate all `sample*.avinput` files:

Option 1 - On Amarel using job arrays ( `submit.sh` file for a 500 sample VCF):

```
#!/bin/bash
#SBATCH --partition=main
#SBATCH --time=2:00:00
#SBATCH --array=0-499
#SBATCH --requeue
inArray=(sample1.avinput sample2.avinput sample3.avinput ...)
input=${inArray[$SLURM_ARRAY_TASK_ID]}
./annotate_variation.pl -build hg19 $input /humandb
```

Option 2 - For loop by (when sample size is small):

```
for f in /path/to/output/folder/sample.*.avinput; do ./annotate_variation.pl -build hg19 $f humandb/; done
```

- Then, calculate *gene score*:

Option 1 - On Amarel using job arrays ( `submit.sh` file refer above).

Option 2 - For loop by:

```
#!/bin/bash
for f in /path/to/sample.*.avinput.exonic_variant_functio
do
  Rscript cal_genescore_make_genescore.R \
    -f $f \
    -s /path/to/db/Mutations.mutOut \
    -l /path/to/db/Transcript-ProtLength_cleaned.csv \
    -m sum \ # or product
    -n both \
    -o /path/to/output/folder
done
```

All gene score calculation functions and pre-processing steps are stored at `cal_genescore_make_genescore.R` script.
`-m` asks user to choose either to sum or multiply variant scores into *gene score*; options: sum, product.
`-n` asks if normalize by protein length; options: y - yes, n - no, both - both.

> Note that, depending on the ANNOVAR humandb version and the NCBI RefSeq version, several genes may have different gene names between the ANNOVAR output (in the `.exonic_variant_function` file) and the `Transcript-ProtLength.csv` file. For example, with an older version of ANNOVAR humandb, transcript NM_138383 mapped to "*MTSS1L*", but to "*MTSS2*" in the NCBI RefSeq database. This happened because the version of ANNOVAR humandb and NCBI RefSeq database aren't 100% exactly the same, or it could be because the gene names are simply not consistent. Therefore, **script `cal_genescore_make_genescore.R` uses transcript NM_ numbers as identifiers, not gene names**. The output *gene score* file contains gene names from the ANNOVAR humandb (currently version 2019Oct24), not NCBI RefSeq.

- Assuming there are 500 individuals in the dataset, 500 resulting files will be generated (*e.g.* `sample.S001.gs` ). Merge them into a data frame where a row is an individual and a column is a gene (protein):

```
Rscript merge_genescore.R \
  -f /path/to/individual/score/folder \
  -o /path/to/output
```

Two files: `GeneScoreTable_normed.txt` and `GeneScoreTable_unnormed.txt` will be created in folder `/path/to/output`. `_normed` means the scores are normalized by protein length.

---

## Step 5: Feature selection (FS) and model building

AVA,Dx by default uses K-S (Kolmogorov–Smirnov) test for FS, random forest for model building, and cross validation to test the predictability of the top-ranking genes. Other FS methods and machine learning models are also included; see below.

- Prepare a cross-validation scheme file ( `cv-scheme.txt` ). For example, we split Tourette's dataset (*e.g.* yale-1 set) into 10 folds and kept all individuals from the same family in the same fold, so that to compare sick *v.s.* healthy instead of differentiating families.

> Cross-validation scheme file ( `cv-scheme.txt` ) should be tab- or comma-separated and contain three columns: *SampleID*, *Phenotype*, *fold*. *Phenotype* **0** means *Negative* and **1** means *Positive*. For example:

```
SampleID fold  Phenotype
sample1 1 1
sample2 1 0
...
sample100 10 1
sample101 10 0
```

- Then, do cross-validation by:

```
Rscript FS-CVperf-kfold.R \
  -f /path/to/GeneScoreTable_normed.txt \
  -m ks \  # current options: ks or DKM
  -M rf \  # current options: rf or SVM
  -s /path/to/cv-scheme.txt \
  -k 10 \
  -l /path/to/Transcript-ProtLength_cleaned.csv \
  -t 5 \  # increase gene numbers by 5
  -n 200 \  # test by top ranked genes until the top 200
  -v 99 \  # 99 means if 99% samples have same scores, the gene is removed
  -o /path/to/output/folder
```

This generates FS results in the output folder. Example outputs are: `10F-CV-ks-selectedGenes.xlsx` and `10F-CV-rf-performance.xlsx` with `-m ks`, `-M rf`, and `-k 10`.

- Then, check pathway over-representation with selected genes. `10F-CV-ks-selectedGenes.xlsx` is needed:

```
Rscript FS-CVgeneOverRep-kfold.R \
  -f /path/to/10F-CV-ks-selectedGenes.xlsx \
  -b /path/to/GeneScoreTable_normed.NAto0.nzv85-15.txt \  # cleaned gene score table
  -n 100 \  # number of genes to select for over-representation analysis
  -d /path/to/CPDB_pathways_genesymbol.tab \
  -a T \  # ascending; T is for KS and F is for CORElearn methods
  -o /path/to/output/folder
```

This generates the over-represented pathways for designated top-ranked genes.

---

## Troubleshooting

- For a larger data frame, *CORElearn* R package may generate error: `Error: protect(): protection stack overflow`. To solve this, set `--max-ppsize=5000000` before running the Rscript; or use `ks` FS method instead.

---

## Supplementary

- Mapping from RefSeq transcript IDs to RefSeq protein IDs can be done via NCBI Batch Entrez. Provide input list of mRNA transcript IDs (e.g. NM_001) and query for protein IDs. This only works for a small amount of queries (< ~2000).

- AVA,Dx has a mapping file from RefSeq transcript IDs to RefSeq protein IDs, and it was extracted from `GCF_000001405.25_GRCh37.p13_rna.gpff.gz` (file date Oct2019) at NCBI ftp on May 6, 2020. The mapping file `GCF_000001405.25_GRCh37.p13_rna.gbff.mRNA_ID-protein_ID.csv` is under the `db` folder. Note that a small part of transcripts (e.g. NM_020452, NM_022375, NM_152666, etc.) in the ANNOVAR reference (2019Oct24) were removed by RefSeq database and will not be retained for further analysis.

- AVA,Dx took the NCBI RefSeq protein database (file date Oct2019) as reference and only kept the longest protein for each gene. When multiple protein isoforms with the same length exist, we only kept the protein ID with the smallest ID number. For example, *AGL* gene has isoforms NP_000019, NP_000633, NP_000634, NP_000635, NP_000637, and all with the same length of 1532 aa. Only NP_000019 was kept for further analysis regardless of whether the isoforms have different sequences.

### Tools:

- To use bcftools plugins: `export BCFTOOLS_PLUGINS=/path/to/bcftools/plugins`

### Amarel:

- To view jobs in detail: `squeue -u ID --format="%.18i %.9P %.18j %.8u %.2t %.10M %.6D %R"`
- Modules: `module load singularity/.2.4-PR1106`, `module load intel/17.0.4 R-Project/3.4.1`
- `scontrol show config`
- `qstat -xml | tr '\n' ' ' | sed 's#<job_list[^>]*>#\n#g' | sed 's#<[^>]*>##g' | grep " " | column -t`