AVA,Dx pipeline

Inputs and outputs

Input:

- VCF file (GRCh37/hg19)
  - e.g. `source.vcf.gz`
- Class label file
  - e.g. `diagnosis.txt`
- (optional) individual list (sample IDs of interest)
  - in case VCF file contains extra individuals
  - e.g. `sampleID_of_interest.txt`
- (optional) cross-validation data split schemes
  - in case of non-random split
  - e.g. `cv-scheme.txt`
- (optional) external gene set to use as features
  - to test the predictability of known genes
  - e.g. `known-genes.txt`

Output:

- a *gene score* table
  - two schemes to choose: *sum* or *product* (see details below)
  - e.g. `GeneScoreTable_normed.txt`
- selected genes
  - e.g. `.xlsx`
- model performance
  - e.g. `performance.xlsx`

Check before running all steps:

- The current workflow works with hg19 only.
- This pipeline is currently for regular VCF file input (scripts need to be updated for gVCF files).
- Manual check of quality outliers, ethnicity, etc. are highly recommended.
- When input VCF contains variants with no SNAP score records in the snapfundb, the snapfun.db needs to be updated.
- Feature selection (FS) and model selection in model training, including FS method choosing, model choosing, model tuning, etc. need human interpretation.

Prerequisite

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

---

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 locations and individuals. Thresholds used here for all metrics are determined empirically. User can change them according to specific need.

- Extract individuals of interest (diseased and healthy individuals of interest).

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

- Remove variant sites which did not pass the VQSR standard.

```
bcftools filter -i 'FILTER="PASS"' source_s-selected.vcf.gz -Oz -o source_s-selected_v-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_s-selected_v-PASS.vcf.gz -Oz -o source_s-selected_v-PASS_snps.vcf.gz
bcftools view --types indels source_s-selected_v-PASS.vcf.gz -Oz -o source_s-selected_v-PASS_indels.vcf.gz
```

- Remove variant sites by site-wise quality. Good site-wise qualities are: QUAL > 30, mean DP > 6, mean DP < 150.

```
bcftools view -i 'QUAL>30 & AVG(FMT/DP)<=150 & AVG(FMT/DP)>=6' source_s-selected_v-PASS_snps.vcf.gz \
  -Oz -o source_s-selected_v-PASS_snps_site-v-Q30-minavgDP6-maxavgDP150.vcf.gz
```

- Check individual call quality. In `filterVCF_by_ABAD.py` : good individual call qualities are: AB > 0.3 and AB < 0.7, GQ > 15, DP > 4; bad individual GTs are converted into missing "./."; low call rate is determined as a call rate < 80%, *i.e.* missing rate >= 20%. Variant sites with a low call rate are removed.

```
python filterVCF_by_ABAD.py \
  source_s-selected_v-PASS_snps_site-v-Q30-minavgDP6-maxavgDP150.vcf.gz \
  source_s-selected_v-PASS_snps_site-v-Q30-minavgDP6-maxavgDP150_gt-v-DP4-AB37-GQ15-MR20perc.vcf.gz
```

- Lastly, gnomAD filter: filtering out variants that were not recorded in the gnomAD database. The gnomAD reference used here is the ANNOVAR gnomAD file `hg19_gnomad_exome.txt` and `hg19_gnomad_genome.txt` . Check the input path of the two reference files before running the script. Note that `tabix` is required for indexing to run this script.

> This step could be slow if there're a lot of variants to check. The gnomAD filter step requires gnomAD variant files `hg19_gnomad_exome_allAFabove0.txt.gz` and `hg19_gnomad_genome_allAFabove0.txt.gz` , which are currently not provided here yet.

```
# 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
```

Note that, 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.

Step 2: Individual QC

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_s-selected_v-PASS_snps_site-v-Q30-minavgDP6-maxavgDP150_gt-v-DP4-AB37-GQ15-MR20perc.vcf.gz > \
  source_s-selected_v-PASS_snps_site-v-Q30-minavgDP6-maxavgDP150_gt-v-DP4-AB37-GQ15-MR20perc.stats.txt
# Draw individual quality figure:
Rscript stats_quality_pca.R \
  -f source_s-selected_v-PASS_snps_site-v-Q30-minavgDP6-maxavgDP150_gt-v-DP4-AB37-GQ15-MR20perc.stats.txt
```

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

> Manual check/interpretation needed for above step.

Ethnicity check:

- Annotate ethnicity with EthSEQ R package:

```
# OPTIONAL: If the number of individuals exceeds certain number, "memory exhausted" error may occur. Manually divide in
bcftools query \
  -l source_s-selected_v-PASS_snps_site-v-Q30-minavgDP6-maxavgDP150_gt-v-DP4-AB37-GQ15-MR20perc.vcf.gz > sample_list.tx
csplit sample_list.txt 500  # outputs from xx00 to xx0n
# OPTIONAL: Clean VCF format for EthSEQ input (do the same thing for every chunk):
bcftools view -S xx00 source_s-selected_v-PASS_snps_site-v-Q30-minavgDP6-maxavgDP150_gt-v-DP4-AB37-GQ15-MR20perc.vcf.gz
  | bcftools annotate --remove 'ID,INFO,FORMAT' \
  | bcftools view --no-header \
  -Oz -o source_xx00_EthSEQinput.vcf.gz
# If no separation of individuals needed:
bcftools annotate --remove 'ID,INFO,FORMAT' \
  source_s-selected_v-PASS_snps_site-v-Q30-minavgDP6-maxavgDP150_gt-v-DP4-AB37-GQ15-MR20perc.vcf.gz \
  | bcftools view --no-header -Oz \
  -o source_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_EthSEQinput.vcf.gz \
  /path/to/output/folder
```

Results of ethnicity predictions are in `/path/to/output/folder/Report.txt` and the corresponding sample IDs are in `sample_list.txt` (`sample_list.txt` obtained by running `bcftools query -l source.vcf.gz > sample_list.txt`).

```
Rscript ethnicity_EthSEQ_summary.R /path/to/output/folder/Report.txt sample_list.txt /path/to/output/folder
```

Above returns two files: `sampleID_closest_EUR.txt` and `sampleID_inside_EUR.txt` . `sampleID_inside_EUR.txt` contains the sample ID for all EUR individuals in the dataset, which generally should be used for further analysis.

Relatedness check:

- Check relatedness within datasets withe the SNPRelate R package. A default kinship > 0.3 is considered to be related.

```
Rscript relatedness.R \
  -i source_s-selected_v-PASS_snps_site-v-Q30-minavgDP6-maxavgDP150_gt-v-DP4-AB37-GQ15-MR20perc.vcf.gz \
  -g source_s-selected_v-PASS_snps_site-v-Q30-minavgDP6-maxavgDP150_gt-v-DP4-AB37-GQ15-MR20perc.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.

Remove individual outliers:

- Outlier individual IDs should be combined from the PCA, ethnicity annotation, and relatedness calculation to a file `outliers.txt` (one ID per row). Then remove individual outliers by:

```
bcftools -S ^outliers.txt \
  source_s-selected_v-PASS_snps_site-v-Q30-minavgDP6-maxavgDP150_gt-v-DP4-AB37-GQ15-MR20perc.vcf.gz -Oz \
  -o source_s-selected_v-PASS_snps_site-v-Q30-minavgDP6-maxavgDP150_gt-v-DP4-AB37-GQ15-MR20perc_ind-cleaned.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_s-selected_v-PASS_snps_site-v-Q30-minavgDP6-maxavgDP150_gt-v-DP4-AB37-GQ15-
# Annotate using hg19 RefSeq:
./annotate_variation.pl -buildver hg19 source_s-selected_v-PASS_snps_site-v-Q30-minavgDP6-maxavgDP150_gt-v-DP4-AB37-GQ1
```

Note that `-format vcf4old` is important to get **all** variants in the VCF. As of May 2020, the latest ANNOVAR version is 2019Oct24. RefSeq is the default reference that ANNOVAR uses. 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 any new dataset, it is very likely that it contains some *new* variants and the `Mutations.mutOut` file does not have their SNAP score. 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:

- Then, 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:
# -e: path to *.exonic_variant_function file
# -m: path to map_RefSeq_and_UniProt.csv file
# -s: path to Mutations.mutOut file (already contains SNAP scores for previous VCF datasets)
Rscript exonic_variant_function2snap_query.R -e source_s-selected_v-PASS_snps_site-v-Q30-minavgDP6-maxavgDP150_gt-v-DP4
```

Note that 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.

But `snapfun.db` is based on UniProt protein records (e.g. P11310 is 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. Query file looks like:

To query for SNAP scores for all variants, including those unmappable ones, we recommend using protein sequences instead of UniProt IDs as query inputs.

- 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
```

The fasta files for SNAP input can be retrieved at NCBI Batch Entrez.

> 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 10-fold cross validation to test the predictability of the top-ranking genes. Other FS methods and machine learning models are also included.

User needs to provide a cross-validation scheme file. For example, we split Tourette dataset (*e.g.* yale-1) into 10 folds and the individuals from the same family enter the same fold, so that to compare sick *v.s.* healthy instead of differentiating families.

- Cross-validation scheme file should be tab-separated and contain three columns: *SampleID*, *Phenotype*, *fold*. 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.text \  # contains columns 'SampleID', 'Phenotype', 'fold'
  -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
  -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 \  # or F depending on the FS method
  -o /path/to/output/folder
```

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

- Permutation test for the significance of the cv performance:

```
# RE-WRITE SCRIPT NEEDED:
Rscript FS-CVperf-kfold-pval~~.R
```

Troubleshooting

- The *CORElearn* package only deals with FS in a relatively small dataset. For a larger data frame, it runs into `Error: protect(): protection stack overflow` and setting `--max-ppsize=5000000` does not help, either.
- **NOT ADDED YET:** The Boruta package has `Boruta()` function, which uses the Boruta algorithm for feature selection. Briefly, Boruta is based on random forest from the ranger package. Boruta gets scores from the random forest ranking of features, and uses shadow features (copies of original features but with randomly mixed values) to keep the feature's original distribution but wipes out its importance. If the original feature has a much higher score compared with its own shadow features, it'll be a *hit*. As a result, Boruta **will** return redundant features.

Original Step 3: Variant annotation and SNAP score calculation

- Run ANNOVAR for VCF annotation:

```
# Convert VCF file into ANNOVAR input format:
convert2annovar.pl -format vcf4old source_s-selected_v-PASS_snps_site-v-Q30-minavgDP6-maxavgDP150_gt-v-DP4-AB37-GQ15-MR
# Annotate using hg19 human reference:
annotate_variation.pl -buildver hg19 source_s-selected_v-PASS_snps_site-v-Q30-minavgDP6-maxavgDP150_gt-v-DP4-AB37-GQ15-
```

- AVA,Dx has pre-calculated SNAP scores stored in the *db* folder (`Mutations.mutOut` ; it has three columns: mRNA accession, amino acid mutation, pre-calculated SNAP score). Additionally in the same folder, `mRNA_identifiers.txt` stores the transcript identifiers (currently 46,327 mRNA transcripts), and `prot_seqs.txt` stores the protein sequences with "NM_XX_prot_NP_YY" in the header lines for mapping uses. For a new dataset, first check if the pre-calculated SNAP score file already has recorded all the variants. If not, we generate a "missing SNAP" list and make SNAP input files for those variants first.

- Check if the dataset has a "missing SNAP score":

```
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
```

If "Transcript-ProtLength.csv lacks protein length info. Check and run this script again.", the script outputs a file of mRNA accession numbers to query at NCBI Batch Entrez, and the `Transcript-ProtLength.csv` file needs to be updated.

- The above command first check if the SNAP score file `Mutations.mutOut` contains all the variants in the dataset. If not, the script prints out the number of "missing SNAP" mutations and generates SNAP input for those variants. To do this, the script first checks if `prot_seqs.txt` file contains all protein sequences and then generates SNAP input files for new variants.

- If `prot_seqs.txt` is not complete, the script outputs a file with mRNA accession numbers (`TranscriptAccess_missing_prot_seq.txt`) in the output folder. User needs to retrieve the corresponding protein sequences at NCBI Batch Entrez.

- NCBI Batch Entrez steps:

    - Upload the mRNA accesion list file with "Choose File" and *Retrieve* in the *Nucleotide database*.
    - Then click *Send to* at the resulting page, choose *Coding Sequences* and *FASTA Protein* to get the protein sequence.
    - Click *Create File* to download.
    - Then, append the protein sequences to the original `prot_seqs.txt` file by:

```
cat prot_seqs.txt sequence.txt > prot_seqs_new.txt
rm prot_seqs.txt
mv prot_seqs_new.txt prot_seqs.txt
```

- Then update the `Transcript-ProtLength.csv` file in *db* folder:

```
Rscript update_Transcript-ProtLength.R /path/to/db/folder
# Check if the output Transcript-ProtLength_update.csv is correct
rm Transcript-ProtLength.csv
mv Transcript-ProtLength_update.csv Transcript-ProtLength.csv
```

Every time if there are new records added o the `Transcript-ProtLength.csv` , user needs to run `clean_Transcript-ProtLength.R` to make sure that the transcripts with the longest protein lengths were kept by:

```
Rscript clean_Transcript-ProtLength.R /path/to/db/Transcript-ProtLength.csv /path/to/db/Transcript-ProtLength_cleaned.c
```

Note that by doing this (above), only one transcript of a gene will be kept. Therefore, if a variant is annotated to a shorter transcript by ANNOVAR, the current pipeline below will just ignore that variant. For example, if a sample has variants chr1:7913029-A-G and chr1:7913445-C-T, the former maps to UTS2:NM_006786:exon1:c.T35C:p.I12T and the latter maps to UTS2:NM_021995:exon1:c.G47A:p.R16Q. We currently only considers the latter for the *gene score* of UTS2 and we ignore the former, because NM_021995 encodes a longer protein than NM_006786 in RefSeq database.

- After the protein sequences are updated by above steps, run the `check_missing_SNAP.R` again to generate mutation files for SNAP input:

```
Rscript check_missing_SNAP.R source_s-selected_v-PASS_snps_site-v-Q30-minavgDP6-maxavgDP150_gt-v-DP4-AB37-GQ15-MR20perc
```

All SNAP input files (*amino acid mutation list* `geneA.mutation` and the *protein fasta file* `geneA.fasta`) will be output to `/path/to/output/folder` , This process might take some time if there are many missing SNAPs.

- After all SNAP input files are ready, SNAP is available on amarel server and can be run using code below (submit.sh SBATCH submission shell script):

```
#!/bin/bash
#SBATCH --partition=bromberg_1,main
#SBATCH --time=72:00:00
#SBATCH --mem=100000
#SBATCH --array=0-999
#SBATCH --requeue
module load singularity/.2.4-PR1106
input=(geneA geneB geneC ...)
singularity exec /home/yw410/bromberglab_predictprotein_yanran-2017-12-06-fa6f97ee098c.img snapfun -i /home/yw410/singu
```

- SNAP outputs plain text files `geneA.out` of predicted variant scores. It needs to be converted to tab-separated format using below code for SNAP output of all genes:

```
#!/bin/bash
for f in /path/to/snap/output/*.out
do
    python snap-scores-mutOut.py $f
done
# Outputs Mutations.mutOut file
```

After all new SNAP output has been converted to tab-separated format, merge them with the original SNAP scores stored in the *db* folder:

```
cat /path/to/db/folder/Mutations.mutOut Mutations.mutOut > /path/to/db/folder/Mutations_new.mutOut
cd /path/to/db/folder/
rm Mutations.mutOut
mv Mutations_new.mutOut Mutations.mutOut
```

Now the *db* folder should have:

- updated SNAP score file `Mutations.mutOut`
- updated transcript - protein - protein length file `Transcript-ProtLength.csv`

Supplementary - other methods for ethnicity check:

- *Method 1*: AIPS from Byun *et al*.

```
# Convert VCF file into plink format.
plink xx
# Merge user file and the reference file.
plink --bfile euro952samples --bmerge input.bed input.bim input.fam --recodeA --out outputA
# Run AIPS-PCA.R and AIPS-AI.R.
```

- *Method 2*: PCA with SNPRelate package in R.

```
Rscript ethnicity_SNPRelate.R \
  source_s-selected_v-PASS_snps_site-v-Q30-minavgDP6-maxavgDP150_gt-v-DP4-AB37-GQ15-MR20perc.vcf.gz
```

- *Method 3*: Calculate probabilities of individuals being a known ethnicity by forensic marker frequency production.

```
# Extract only the 55 markers from KiddLab.
bcftools view -R 55markers.txt source_s-selected_v-PASS_snps_site-v-Q30-minavgDP6-maxavgDP150_gt-v-DP4-AB37-GQ15-MR20pe
# Calculate the probability using a production method.
Rscript forensic_method.R source_Kidd-markers.vcf.gz
```

- **Note that:**
    - *Method 1* uses AIMs to infer ethnicity using reference labels (952 ancestry known samples).
    - *Method 2* takes all SNPs and do PCA on LD-pruned SNPs to infer ethnicity using reference labels (user defined).
    - *Method 3* uses AIMs to infer ethnicities (known ethnicities).
    - Technically, results should be very **consistent across all method**. But human interpretation may be needed for specific cases.

RefSeq GRCh37 mRNA and protein sequences downloaded on May/5/2020.

ANNOVAR humandb hg19_refGeneMrna.fa with a date on June/1/2017.