DVAR is a Bayesian framework to assess functional consequences of noncoding variants through annotation data integration. It takes noncoding variants list as the input and generated functional cluster labels and scores for each variant. In general, the framework is divided into two levels: The underlying DVAR-Computing component (include DVAR-Cluster and DVAR-Score) and high-level DVAR-Command component (support command input).

The underlying DVAR-Computing component is developed with C++ program language. The source codes of them are available at ./code/DVAR/. The DVAR-Computing module requires detailed parameters and the configuration file to construct Bayesian model:

Command options:	
-v [--version]	Version number
--help	Usage tips
-c [--config](=DVAR.cfg)	Configure file name
Configuration or Command options:	
--ThreadNum(=8)	Number of CPU threads
--Model(=DVAR)	Model name, only support DVAR
--Run_Mode(=cluster)	Run mode used in DVAR-Cluster
--VecSize(=0)	Dimension of input data
--TruncateNum(=0)	Used for model training. DVAR use TruncateNum=8
--FeatureList	Feature input list
--LabelFile	label list of the training variants
--Weight_min(=0.0001)	The minimum of weight of a mixture
--Hp_Path(=./output)	Model file save directory
--beta0(=0.01)	Constant
--beta1(=0)	Constant
--alpha_s1(=500)	Constant
--alpha_s2(=1)	Constant
--Nu0(=10)	Constant
--lambda1(=1)	Scoring algorithm weight coefficient
--lambda2(=4)	Scoring algorithm weight coefficient
--Score_Result(=result.score)	Score results
--Cluster_Result(=result.cluster)	Cluster results
--Em_Iternum_Max	The number of training iterations
--Fast_Threshold (=0.01)	The threshold of the early stop technology

Depends on the high efficiency and simplicity of C++, we can generate the functional scores of noncoding variants across the whole genome with a short time. The following libraries are also required to compile the source code:

Intel math kernel library (MKL>=V10.1) https://software.intel.com/en-us/mkl
Armadillo linear algebra library (>=V7.800.3) http://arma.sourceforge.net/
c++ boost library (>=V1.58) http://www.boost.org/
GNU Scientific Library (GSL) (>=V1.13) https://www.gnu.org/software/gsl/

DVAR-Computing support to be compiled with c++ compilers which supported C++ 11 standard and openMP

(multi-threading support). The sources codes package also supports automatic compile and generated executables with Cmake (>=V3.6) while the paths of the dependencies packages have been specified. We tested and recommended the compiler environment: GCC 4.8.5 and Cmake 3.6.3 on Centos 6.8. Users can use the command "cmake -DCMAKE_BUILD_TYPE=Release ./" at ./code/DVAR/ path to generate the makefile, then use the command 'make install' to generate binary file at ./model. All the required libraries should be installed and the Cmake list file CMakeLists.txt at ./code/DVAR/ should be modified to identify the libraries. The MKL package can be found at anaconda pkg path if anaconda is installed.

DVAR-Computing support three running modes: model-train (model learning); model-score (score); model-cluster (cluster). Users do not need to fall into the details of how to use DVAR since we have the high-level component DVAR-Command which can call the DVAR-Computing component internally.

The DVAR-Command is developed with python 2.7. The following libraries are also required to run the script: h5py, numpy, statsmodels, pandas and sklearn. We suggest the user install anaconda2 to get most of the required libraries such as Intel MKL (https://anaconda.org/anaconda/python).

The input of the DVAR component can be a list of noncoding variants like:

```
chr1  3691528    A  G  rs1175550
chr1653800954  T  C  rs1421085
chr1917393925  C  A  rs56069439
chr1930303380  A  -  rs200996365
chr2055990405  T  C  rs737092
chr3  37034946   G  A  rs1800734
chr4  111553133  T  G  rs2595104
chr6  109625879  G  A  rs1546723
chr6  117210052  T  C  rs339331
```

On each line, the tab-delimited columns represent chromosome, position, the reference nucleotides, the observed nucleotides, and the rs number.

Then, based on the variant list, DVAR extract features of each variant and saved in: /fea. We provide the feature files of the training data, clustering testing data and scoring testing data for the reproduction of the study of 'De Novo pattern discovery enables robust assessment of functional consequences of noncoding variants'. All of the functional clusters and scores pre-computed by DVAR across the whole genome-region can be found at: /hg19. For example, users can use 'tabix' to extract the cluster label and score of rs1175550 (chr1: 3691528) from the pre-computed files (cluster label file:hg19_DVAR.cluster.gz; score file: hg19_DVAR.score.gz):

```
tabix ./hg19/hg19_DVAR.cluster.gz 1:3691528-3691528
tabix ./hg19/hg19_DVAR.score.gz 1:3691528-3691528
```

The data structure of DVAR feature is shown below:

	chr	pos	annotation0	annotation1	annotation2	annotation3
v0	1	3691528	7.619100	2.109254	0.431056	-2.671226
v1	16	53800954	3.162452	1.294150	1.334790	-3.612958
v2	19	17393925	14.586820	2.850681	-3.005013	3.486832
v3	19	30303380	13.465409	4.227807	0.567969	3.844155
v4	20	55990405	3.683647	1.503699	-0.830125	-1.566496
			...			

On each line, the tab-delimited columns represent chromosome, position, the annotations features 0, 1, 2, 3.
The feature data is a pandas data frame object. We save and load of the data with the function pandas.to_hdf() and pandas.read_hdf().

DVAR (python) support three running modes: model-train (train); model-score (score); model-cluster (cluster).
While the user needs to train the Dirichlet Process Mixture model based on the training data, the list of noncoding vairants and feature data need to be provided with the same format as we shown in ./input/train.input and ./fea/train.fea. Then run the command in the shell:

```
python DVAR.py -m train -i ./input/train.input
```

Similarly, if the user needs to get the clustering label, the list of noncoding variants and feature data need to be provided with the same format as we shown in ./input/examples.input and ./fea/examples.fea. Then run the command in the shell:

```
python DVAR.py -m cluster -i ./input/examples.input
```

The output file is ./score/examples.cluster. The user can also get the functional scores. The list of noncoding variants and feature data need to be provide with same format as we shown in: ./input/examples.input and ./fea/examples.fea. Then run the command in the shell:

```
python DVAR.py -m score -i ./input/examples.input
```

The output file is ./score/examples.score. Note that the training and testing process needs to be taken together unless using the input file provided by us.

The contents of the folder are as follows:

Folder structure: /	
/code/	Source code of DVAR
/fea/	Feature data
/hg19/	Pre-computed DVAR-cluster labels and DVAR scores across the whole genome
/input/	Input of the variants list
/model/	The DVAR model

/score/	The DVAR scores and cluster labels