VARITY FRAMWORK USER GUIDE

1. Introduction

VARITY is a supervised machine learning approach to build specialized predictive models using training data with optimized differential weights. VARITY allows different weights to be placed on different training examples. A high-quality core set of examples are given full weight, while examples from diverse add-on sets with potentially lower predictive utility are subjected to filtering and weighting. For each add-on set, examples are ordered by one or more quality-informative properties, and a threshold used to filter out the examples with low predictive utility, with a single weight assigned to all retained examples. Filtering thresholds and weights are treated as hyper-parameters and optimized for performance on the core set of examples using cross-validation.

1.1 Steps to apply VARITY framework

1) Assemble training data (both high-quality core set and add-on sets with uncertain quality).
2) Optional step: Identify quality-informative properties for each add-on set using moving window analysis.
3) Config filtering and weighting hyperparameters for each add-on set.
4) Run hyperparameter optimization to determine filtering thresholds and weight for each add-on set.

5) Train the final VARITY model with core set and weighted add-on sets.

1.2 To build VARITY models

1) Download [VARITY project folder] (http://varity.varianteffect.org/downloads/VARITY_Final.zip) and extract to a local folder (as your project folder) to keep all raw data and results. The downloaded folder contains an existing session named "Final" for the manuscript ***"Improved pathogenicity prediction for rare human missense variants"***. Using [VARITY python scripts] (https://github.com/joewuca/varity/tree/master/python), you can explore the existing session or create a new session to build your own VARITY models. Please read rest of this user guide before you process to next step.
2) Explore "Final" session for VARITY_R and VARITY_ER models. Example commands are in [Final session commands] (https://github.com/joewuca/varity/tree/master/VARITY_Final_session_commands.txt).
3) Create a new session to build your own VARITY models. Example commands are in [new session commands] (https://github.com/joewuca/varity/tree/master/VARITY_new_session_commands.txt).

1.3 System Requirement

Python version 3.7.2 and a few python packages listed in Table 1.

1.4 Technical Support

Please contact joe.wu.ca@gmail.com for technical support.

2. VARITY data preparation

All core and add-on sets training examples need to be assembled into one single CSV file as an input, and data columns include:

- Feature columns (currently only supports Real or Integer type features)
- A "label" column: the dependent variable (0 or 1 for binary classification problem)
- A "extra_data" column: 0 for core examples, and 1 for add-on examples.
- A "extra_data_name" column: name for different add-on sets.
- Candidate quality-informative property columns.

VARITY data files location: */project_path/data*

3. VARITY session configuration

VARITY session uses a configuration file (*/project_path/config/YOUR_SESSION_ID.vsc)* to setup everything needed for training a VARITY model. The configuration file contains settings for four different type of objects used in VARITY framework: 1) Data 2) Estimator 3) Predictor 4) Hyperparameter. Each object has a list of attributes, and their names can be found in a “object definition” line (starts with “*” symbol) delimited by “|” sign. A new object instance can be created below the “object definition” line, starting with the object instance name and then values of each attribute. A valid session config file is needed before running any VARITY commands. One can use the provided config file (*/project_path/config/* varity_config.vsc) as a template and rename to your own session id.

3.1 Data object configuration

The data object takes the input training data (consists of core and add-on training examples) and splits it in nested cross validation fashion. The core training examples were first splitted into *k1* outer-loop folds. For each outer-loop, the core training examples are splitted into an outer-loop training set and an outer-loop test set, and subsequently the outer-loop training examples are further splitted into *k2* inner-loop folds. The purpose of outer-loop cross-validation is to fairly evaluate model performance on core set of examples, and inner-loop cross-validation is to optimize hyperparameter for each outer-loop. The attributes for data object:

- **test_split_method**

The way of splitting for outer-loop cross-validation.

0: random split

1: stratified split (keep same prior for the training and test in each outer-loop fold)

- **test_split_folds**

Number of folds (*k1)* for outer-loop cross-validation

- **test_split_ratio**

Only used when the “test_split_folds” attribute is set to 1. When there is only one outer-loop fold, this value determines the fraction of the core set of examples as test set. If the value is set to 0, the corresponding inner-loop cross-validation will be based on the whole core set of examples, which is the appropriate configuration for the data object used for building a final VARITY model.

- **cv_split_method**

The way of splitting for inner-loop cross-validation.

0: random split

1: stratified split (keep same prior for the training and validation in each inner-loop fold)

- **cv_split_folds**

Number of folds for inner-loop cross-validation

- **cv_split_ratio**

Only used when the “cv_split_folds” attribute is set to 1. When there is only one inner-loop fold, this value determines the fraction of the outer-loop training examples as inner-loop validation set.

- **data_file**

The absolute path of the input training data.

3.2 Estimator object configuration

VARITY framework is designed to support a list of different machine learning algorithms. However, currently it only support the gradient boosted trees algorithm. The attributes for data object:

- **algo_name**

The name of the learning algorithm. The name for gradient boosted trees for binary classification is “xgb_c”

- **round_digits**

The significant digits of the output metrics (e.g., AUROC)

3.3 Predictor object configuration

The predictor object is associated with one data object instance and one estimator instance, and has following attributes:

- **type**

0: VARITY model predictor

1: Other existing predictor (predictions can be found as one column in the data file)

- **ml_type**

VARITY framework is designed to support different type of machine learning tasks. However, currently only the following type is supported:

“classification_binary”: binary classification

- **data**

the name of the associated data object instance

- **estimator**

the name of the associated estimator object instance

- **tune_obj**

VARITY framework is designed to support a list of metrics as the objective function for hyperparameter optimization and performance evaluation using cross-validation. Currently only the following metrics are supported:

“macro_cv_aubprc”: The Area Under Balanced Precision Recall Curve via cross-validation

“macro_cv_auroc”: The Area Under ROC curve via cross-validation

- **hyperopt_trials**

The number of hyperparameter tuning trials (HyperOpt trials)

- **trials_mv_size**

The number of trials in one moving window (used for select the best trial, see Section 4 save_best_hp command) .

- **shap_test_interaction**

0: disable

1: enable output feature contribution (SHAPley values) for each prediction when run “target_prediction” command to predict target data.

- **shap_train_interaction**

0: disable

1: enable output feature contribution (SHAPley values) for each training example when run “target_prediction” command to predict target data.

- **features**

A list of features used for the predictor. For other predictors with existing predictions (type = 1), just use the corresponding column name in the data file as a single feature. All features need to be put in square brackets ([]) and separated by comma.

3.4 Hyperparameter configuration

There are two types of hyperparameters for a VARITY predictor; 1) algorithm level parameters with respect to the estimator. 2) filtering and weighting parameters for add-on sets. For each add-on set, only one weighting parameter but multiple filtering parameters are allowed. Each hyperparameter has following attributes:

- **predictor**

  It is possible one hyperparameter used for different predictors (use square bracket to include all predictors separated by comma).

- **hp_type**

1: filtering parameter

2: weighting parameter

3: algorithm parameter

- **source**

The associated list of add-on sets (only for filtering and weighting hyperparameters). A valid add-on set name is one of the distinct values in the "extra_data_name" column of the training data. All add-on set name need to be put in square brackets ([]) and separated by comma.

- **orderby**

  The informative property used for ordering (in ascending) the examples for the associated add-on set ('source' attribute). A valid informative property name is one of the column names of the training data.

- **from**
  The lower bound of the hyperparameter value.

- **to**

  The higher bound of the hyperparameter value. For filtering parameters, the higher bound is determined automatically based on number of examples in the associated add-on set.

- **step**

  The difference between each hyperparameter value from the lower bound to higher bound. For filtering parameters, this parameter indicates the number of examples in a "filtering block". The add-on set is filtered out in blocks instead of individual example.

- **direction**

  The parameter is only relevant to filtering parameter:

0: Filtering out the add-on set from the examples with lowest informative property value.

1: Filtering out the add-on set from the examples with highest informative property value.

- **default**

  The default value of the hyperparameter.

- **type**

  The data type of the hyperparameter (int or real)

- **mv_size_precent**

  For moving window analysis, this parameter determines the number of examples in each moving window, which equals to product of [number of examples in an add-on set] and [mv_size_percent] (rounded).

- **mv_data_points**

  Number of moving windows for moving window analysis

- **enable**

  1: The hyperparameter is enabled for hyper-parameter tuning

  0: The hyperparameter is disabled for hyper-parameter tuning

4. VARITY commands

To run VARITY commands, please take the following steps first:

1) Download code in this git repository (https:/github/jowuca/varity/VARITY_exmaple.zip) to a local folder as your VARITY script folder.
2) Make sure you have installed python 3, and all associated packages needed for VARITY framework (See Table 1)
3) Make sure your python PATH has included the VARITY script folder.

VARITY framework currently supports the following commands:

- **init_session**

  python3 varity_run.py ***actions****=init_session* ***session_id****=YOUR_SESSION_ID* ***project_path****=PATH_OF_YOUR_PROJECT_FOLDER*

  This command initiates a session (creating all necessary VARITY framework objects) based on the current configuration file. Unless you reinitiate the session again, changes to the configuration file will not affect the initialized VARITY objects. You can reinitiate the session by adding argument ***reinitiate****=1* to the command line, but you might need to generate all your results again after session re-initiation.

output:

- */project_path/output/npy/[**session_id**]_[EACH DATA_INSTANCE_NAME]_savedata.npy*
- */project_path/output/npy/[**session_id**]_[EACH PREDICTOR_INSTANCE_NAME]_hp_config_dict.npy*

* **mv_analysis**

  *python3 varity_run.py **actions**=mv_analysis **session_id**=YOUR_SESSION_ID **predictor**=PREDICTOR_INSTANCE_NAME **filtering_hp**=HYPERPARAMETER_INSTANCE_NAME **project_path**=PATH_OF_YOUR_PROJECT_FOLDER*

  Moving analysis first order the add-on set ("source" attribute of ***filtering_hp***) examples by the informative property ("orderby" attribute of ***filtering_hp***) in ascending or descending order ("direction" attribute of ***filtering_hp***), then create ["mv_data_points" attribute of ***filtering_hp***] number of moving windows (size of each window equals to [number of examples in add-on set] * ["mv_size_precent" attribute of ***filtering_hp***]). The predictive utility of each window is estimated using 10-fold cross validation on the core training set, where the training examples in each fold were supplemented by all of the add-on examples in that moving window.

  output:

  - */project_path/output/csv/[**session_id**]_[**predictor**]_[**filtering_hp**]_spvalue_results.txt*

* **plot_mv_result**

  *python3 varity_run.py **action**=plot_mv_result **session_id**=YOUR_SESSION_ID **predictor**= PREDICTOR_INSTANCE_NAME **filtering_hp**=HYPERPARAMETER_INSTANCE_NAME **project_path**=PATH_OF_YOUR_PROJECT_FOLDER*

  Plot the moving analysis result (the predictive utility of each moving window)

  output:

  - */project_path/output/img/[**session_id**]_[**predictor**]_[**filtering_hp**]_mv_result.png*

* **hp_tuning**

  *python3 varity_run.py **action**=hp_tuning **session_id**=YOUR SESSION ID **predictor**=PREDICTOR_INSTANCE_NAME **cur_test_fold**= THE OUTER-LOOP FOLD **project_path**=PATH_OF_YOUR_PROJECT_FOLDER*

Hyperparameter tuning for the input ***predictor*** on the specified outer-loop fold. For the predictors for nested cross validation, the possible outer-loop fold value is from 0 to “test_split_folds” attribute of the corresponding data object minus one. For the final VARITY model predictor (compare to predictors used for nested cross-validation), there is only one dummy outer loop therefore the outer-loop fold should be set to 0.

output:

- */project_path/output/npy/[**session_id**]_[**predictor**]_[**filteing_hp**]_tf[**cur_test_fold**]_trials.pkl*
- */project_path/output/csv/[**session_id**]_[**predictor**]_[**filtering_hp**]_tf[**cur_test_fold**]_trial_results.txt*

- **save_best_hp**

  *python3 varity_run.py **action**=save_best_hp **session_id**=YOUR_SESSION_ID **predictor**=PREDICTOR_INSTANCE_NAME **cur_test_fold**=THE OUTER-LOOP FOLD **project_path**=PATH_OF_YOUR_PROJECT_FOLDER*

  Select and save the optimum hyperparameter setting from all hyperparameter optimization (HyperOpt) trials using the following procedure: 1) Re-order all trials by mean metric (“tune_obj” attribute of ***predictor***) on training sets (averaged over 10 training sets) from low to high; 2) calculate a moving window (we used window size 100) average of mean metric on validation sets; 3) define an “early stopping” point at the first moving window (the “fittest” region) for which mean metric on validation sets begins to descend; 4) Select as final the hyperparameters from the trial within this “fittest” region that achieved the highest mean metric on validation sets.

  output:

  - */project_path/output/npy/[**session_id**]_[**predictor**]_tf[**cur_test_fold**]_hp_dict.npy*
  - */project_path/output/csv/[**session_id**]_[**predictor**]_tf[**cur_test_fold**]_best_hps.csv*
  - */project_path/output/img/[**session_id**]_[**predictor**]_tf[**cur_test_fold**]_hp_selection.png*

- **plot_hp_weight**

  *python3 varity_run.py **action**=plot_hp_weight **session_id**=YOUR_SESSION_ID **predictor**=PREDICTOR_INSTANCE_NAME **cur_test_fold**=THE OUTER-LOOP FOLD **filtering_hp**=HYPERPARAMETER_INSTANCE_NAME*

  This command plots the weight of each example in an add-on set or combined add-on sets (“source” attribute of ***filtering_hp***) ordered by the informative property (“orderby” of

filtering_hp). The filtering threshold and weight are based on the optimized hyperparameters.
output:

- */project_path/output/img/[**session_id**]_[**predictor**]_[**filtering_hp**].png*

- **test_cv_prediction**

  *python3 varity_run.py **action**=test_cv_prediction **session_id**=YOUR_SESSION_ID **predictor**=PREDICTOR_INSTANCE_NAME*

  This command runs nested cross-validation. For each outer-loop, It makes predictions the test set using model trained with the optimized hyper-parameters via inner-loop cross-validation.
  output:

  - */project_path/output/npy/[**session_id**]_[**predictor**]_test_cv_results.npy*
  - */project_path/output/csv/[**session_id**]_[**predictor**]_hp_test_cv_results.csv*

- **plot_test_result**

  *"python3 varity_run.py **action**=plot_test_result **session_id**=YOUR_SESSION_ID **predictor**= PREDICTOR_INSTANCE_NAME **compare_predictors**=[PREDICTORS_FOR_COMPARISON]*

  Plot the balanced precision recall curve and ROC curve using the results from nested cross-validation. The statistical test is carried out between each predictor specified in ***compare_predictors*** (usually non-VARITY predictors) and the predictor specified in ***predictor*** (usually VARITY predictor). For each outer-loop fold, the test set is filtered if there is a missing prediction from any of the predictor specified in ***compare_predictors***.
  output:

  - */project_path/output/npy/[**session_id**]_[**predictor**]_filter_1_auroc_interp.png*
  - */project_path/output/csv/[**session_id**]_[**predictor**]_ filter_1_aubprc_interp.png*

- **targe_prediction**

  *python3 varity_run.py **action**=target_prediction **session_id**=YOUR_SESSION_ID **predictor**= PREDICTOR_INSTANCE_NAME **cur_test_fold**=THE OUTER-LOOP FOLD **target_file**=TARGET_FILE_NAME **loo**= [0 OR 1]*

  Predict the examples in ***target_file***. If the ***loo*** is set to 1, then only the target examples that have been used in training will be predicted using leave-one-example-out strategy.
  output:

- */project_path/output/npy/[**session_id**]_[**predictor**_tf[**cur_test_fold**]_target_predicted.csv*
- */project_path/output/npy/[**session_id**]_[**predictor**_tf[**cur_test_fold**]_target_loo_predicted.csv*

5. Table 1: Required python packages

Package Name	Version	Description (Link to document)
Cython	0.29.14	C extension for python (https://cython.org/)
graphviz	0.13.2	Open source graph visualization software (https://graphviz.org/)
hyperopt	0.2.2	Bayesian hyperparameter optimization (https://github.com/hyperopt/hyperopt)
matplotlib	3.1.0	Python visualization (https://matplotlib.org/)
numpy	1.16.0	Scientific computing with Python (https://numpy.org/)
pandas	0.24.0	Data analysis and manipulation tool (https://pandas.pydata.org)
scikit-learn	0.20.2	Machine learning in Python (https://scikit-learn.org/stable/)
scipy	1.2.0	Python-based ecosystem of open-source software for mathematics, science, and engineering (https://www.scipy.org/)
seaborn	0.9.0	Statistical data visualization (https://seaborn.pydata.org/)
shap	0.34.0	A game theoretic approach to explain the output of any machine learning model (https://github.com/slundberg/shap)
xgboost	0.90	An optimized distributed gradient boosting library (https://xgboost.readthedocs.io/en/latest/)