DriverPower Documentation

Release 1.0.0

Shimin Shuai

May 29, 2020

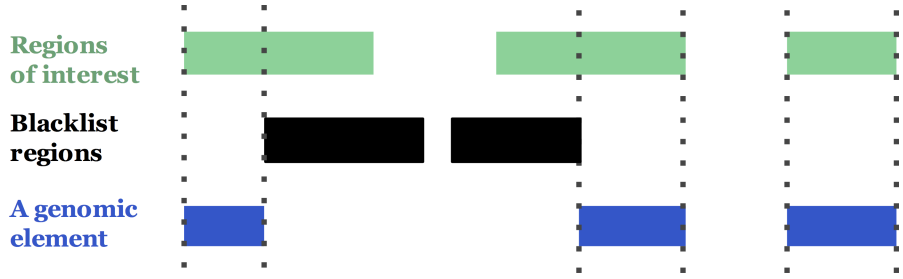
Documentation

CHAPTER 1

Tutorial with example data

DriverPower aims to identify cancer driver candidates from **somatic** variants of a tumour cohort. The minimal number of samples required in the PCAWG study is 15. However, more samples will have more power to detect rare driver events. Both whole-genome sequencing and panel sequencing variants can be used.

The basic test unit accepted by DriverPower is called a **genomic element**, which is a set of disjoint (or continuous) genomic regions (see the figure below). In our manuscript, we also removed all blacklist regions to reduce the effect of variant artifacts. In the real world, a genomic element can be all exons of a gene, or all TF binding sites among a promoter etc.



Typically 1K to 100K genomic elements are tested together such as ~20K genes in the human genome. For each element, given its functional impact, mutational burden as well as thousands of features, DriverPower will report a p-value and q-value associated with that element.

1.1 0: Download example data

Our example data are hosted on figshare, on which you can find the following files:

1. random_mutations.tsv.gz [14 MB]: randomized whole-genome somatic mutations from 50 patients.
2. train_feature.hdf5.part1 [4 GB]
3. train_feature.hdf5.part2 [4 GB]
4. train_feature.hdf5.part3 [900 MB]

5. test_feature.hdf5 [1.4 GB]
6. train_elements.tsv.gz [7 MB]
7. test_elements.bed12.gz [6 MB]
8. callable.bed.gz [5 MB]: whitelisted genomic regions used in DriverPower; can be substituted with a chromosome length file (BED format) to use the entire genome.

Important: You can run DriverPower for your own data by simply replacing **random_mutations.tsv.gz** with your mutations.

The training features has been divided into three parts, so you will need to merge them:

```
cat train_feature.hdf5.part1 train_feature.hdf5.part2 train_feature.hdf5.part3 >␣
↪train_feature.hdf5
md5sum train_feature.hdf5  # should be cb4af6bc7979efa087955f94909dd065
```

1.2 1: Install Python and DriverPower

For this tutorial, we used a brand new virtual machine with 15 CPUs and 125GB RAM. The OS we used is Ubuntu 18.04.1 LTS (GNU/Linux 4.15.0-33-generic x86_64).

To install Python3 using Anaconda3 if you don't already have:

```
# Find the current version from Anaconda (https://repo.anaconda.com/archive/)
wget https://repo.anaconda.com/archive/Anaconda3-[version]-Linux-x86_64.sh
# You may need sudo to run this
./Anaconda3-[version]-Linux-x86_64.sh
. .bashrc
```

It's always better to create a new conda environment, so that all your existing packages won't get messed up:

```
# [Optional] Create a new environment for DriverPower
conda create --name driverpower
# [Optional] Activate the environment
conda activate driverpower
```

Then we can install required packages and DriverPower:

```
# Setup Bioconda Channels if not set before
# https://bioconda.github.io/user/install.html#set-up-channels
conda config --add channels defaults
conda config --add channels bioconda
conda config --add channels conda-forge
# Install DriverPower
conda install -c smshuai driverpower
```

1.3 2: Make response tables

The response (y; dependent variable) table records the observed number of mutations, number of mutated samples and the length per genomic element. This table is required for both `model` (training) and `infer` (test) sub-commands.

You can make them easily using our helper script `prepare.py`. Inputs will be mutations and elements:

```
# Get the helper
wget https://raw.githubusercontent.com/smshuai/DriverPower/master/script/prepare.py
# Training responses
python ./prepare.py random_mutations.tsv.gz train_elements.tsv.gz callable.bed.gz␣
↪train_y.tsv
# Test responses
bedtools bed12tobed6 -i ./test_elements.bed12.gz  | cut -f1-4 > test_elements.tsv
python ./prepare.py random_mutations.tsv.gz test_elements.tsv callable.bed.gz test_y.
↪tsv
```

1.4 3: Build the background mutation rate model

The background mutation rate (BMR) model is used to estimate the expected number of somatic mutations for each genomic element, given its features. DriverPower sub-command `model` is used to train BMR models. To build the BMR model, training features (X) and responses (y) are required. DriverPower supports two algorithms for the BMR model, generalized linear models (GLM) and gradient boosting machines (GBM).

Here we show how to build a GBM with our example data:

```
mkdir output
driverpower model \
    --feature train_feature.hdf5 \
    --response train_y.tsv \
    --method GBM \
    --name tutorial \
    --modelDir ./output
```

You should see the following log info (taking ~3 hours on our VM):

```
09/08/2018 20:42:37 | INFO: Welcome to DriverPower v1.0.1
09/08/2018 20:42:59 | INFO: Successfully load 1373 features for 867266 bins
09/08/2018 20:43:05 | INFO: Use 867266 bins in model training
09/08/2018 20:43:14 | INFO: Split data fold 1/3
09/08/2018 20:43:19 | INFO: Split data fold 2/3
09/08/2018 20:43:25 | INFO: Split data fold 3/3
09/08/2018 20:43:31 | INFO: Training GBM fold 1/3
[0]     eval-poisson-nloglik:114024
Will train until eval-poisson-nloglik hasn't improved in 5 rounds.
[100]   eval-poisson-nloglik:25279.4
.......omit many lines.............
Stopping. Best iteration:
[1128]  eval-poisson-nloglik:1.38992

09/08/2018 23:56:35 | INFO: Model metrics for training set: r2=0.63, Variance␣
↪explained=0.63, Pearson'r=0.79
09/08/2018 23:56:42 | INFO: Job done!
```

1.5 4: Infer driver candidates

DriverPower can be used to find driver candidates with or without functional information. This step will use the model file `./output/tutorial.GBM.model.pkl` from last step.

We first show how to call driver candidates **without** functional information, aka, burden-test only:

```
driverpower infer \
    --feature test_feature.hdf5 \
    --response test_y.tsv \
    --model ./output/tutorial.GBM.model.pkl \
    --name 'DriverPower_burden' \
    --outDir ./output/
```

To use functional information, one or more types of functional measurements (e.g., CADD, EIGEN, LINSIGHT etc) need to be collected first. The CADD scores can be retrieved via its web interface (up tp 100K variants each time) without downloading the large file for all possible SNVs (~80 G). If you have more than 100K variants, you can either split your file and run the web app multiple times, or download the large file and try `tabix`. Other scores can be obtained using a similar method after download. After obtaining the per-mutation score, you can calculate the average score per element, which will be used by DriverPower.

Here we show how to score 1,000 mutations and calculate per-element score:

```
# We omit INDELs here; but CADD can score INDELs in VCF format
zcat ./random_mutations.tsv.gz | \
awk 'BEGIN{OFS="\t"} $4 != "-" && $5 != "-" {print $1,$3,".",$4,$5}' | \
head -1000 | gzip > random_mutations.1K.vcf.gz
# Upload formatted variants (random_mutations.1K.vcf.gz) to CADD's web interface
# and download the result file (something like GRCh37-v1.4_
↪f8600bd0c0aa23d4f6abc99eb8201222.tsv.gz).
#####
# Intersect the score file (we use the PHRED score) with test elements
zcat ./GRCh37-v1.4_f8600bd0c0aa23d4f6abc99eb8201222.tsv.gz | \
tail -n +3 | awk 'BEGIN {OFS="\t"} {print "chr"$1, $2-1, $2, $6}' | \
bedtools intersect -a ./test_elements.tsv -b stdin -wa -wb > CADD_ele.tsv
# The 4th column is the element ID and the 8th column is the CADD PHRED score
printf "binID\tCADD\n" > CADD_per_ele_score.tsv
bedtools groupby -i ./CADD_ele.tsv -g 4 -c 8 -o mean  >> CADD_per_ele_score.tsv
```

We can now supply the per-element score file to DriverPower and call driver candidates:

```
driverpower infer \
    --feature test_feature.hdf5 \
    --response test_y.tsv \
    --model ./output/tutorial.GBM.model.pkl \
    --name 'DriverPower_burden_function' \
    --outDir ./output/ \
    --funcScore CADD_per_ele_score.tsv \
    --funcScoreCut "CADD:0.01"
```

1.6 5: Misc.

TODO

CHAPTER 2

Installation guide

DriverPower requires Python >= 3.5 and some other computing packages. If you don't have Python 3.5 or higher, we recommend to install Python with Anaconda.

2.1 Install from conda

The best way to install driverpower is to use `conda`:

```
# Setup bioconda channels (see https://bioconda.github.io/user/install.html#set-up-
↪channels)
conda config --add channels defaults
conda config --add channels bioconda
conda config --add channels conda-forge

# Create a new env and install DriverPower
conda create -n driverpower
conda activate driverpower
conda install -c smshuai driverpower
```

2.2 Install from pypi or source

2.2.1 Install from pypi

You can install DriverPower from the Python Package Index (PyPI) by simply typing the following command in your terminal:

```
$ pip install driverpower
```

2.2.2 Install from source

You can also download the latest source from GitHub to install. For example, to install version 1.0.x (change x to the right version number):

```
$ tar -xzf DriverPower-1.0.x.tar.gz
$ cd DriverPower-1.0.x/ && pip install .
```

CHAPTER 3

Command-line interface

DriverPower uses a command-line interface. To view available sub-commands and usage of DriverPower, type

```
$ driverpower -h
usage: driverpower [-h] [-v] {model,infer} ...

DriverPower v1.0.0: Combined burden and functional impact tests for coding
and non-coding cancer driver discovery

optional arguments:
  -h, --help     show this help message and exit
  -v, --version  Print the version of DriverPower

The DriverPower sub-commands include:
  {model,infer}
    model        Build the background mutation model
    infer        Infer driver elements
```

3.1 The `model` sub-command

The `model` sub-command is used to build the background mutation model with training data. Command-line options for this command can be viewed as follows:

```
$ driverpower model -h
usage: driverpower model [-h] --feature str --response str [--featImp str]
                         --method {GLM,GBM} [--featImpCut float]
                         [--gbmParam str] [--gbmFold int] [--name str]
                         [--modelDir str]

optional arguments:
  -h, --help          show this help message and exit

input data:
```

(continues on next page)

(continued from previous page)

```
  --feature str       Path to the training feature table (default: None)
  --response str      Path to the training response table (default: None)
  --featImp str       Path to the feature importance table [optional]
                      (default: None)

parameters:
  --method {GLM,GBM}  Algorithms to use (default: None)
  --featImpCut float  Cutoff of feature importance score [optional] (default:
                      0.5)
  --gbmParam str      Path to the parameter pickle [optional] (default: None)
  --gbmFold int       Train gbm with k-fold, k>=2 [optional] (default: 3)
  --name str          Identifier for output files [optional] (default:
                      DriverPower)
  --modelDir str      Directory of output model files [optional] (default:
                      ./output/)
```

- **Input**
  - Required data: `--feature` and `--response` for training elements.
  - Optional data: `--featImp`.
- **Output**
  - all output files are in `--modelDir`.
  - `[name].GLM.model.pkl`: the GLM model file.
  - `[name].GBM.model.fold[k]`: the GBM model files.
  - `[name].[GLM|GBM].model_info.pkl`: the corresponding GLM or GBM model information.
  - `[name].feature_importance.tsv`: the feature importance table, which is returned when no input `--featImp`. For GLM, feature importance is the number of times a feature is used by randomized lasso. For GBM, feature importance is the average gain of the feature across all gradient boosting trees.

Important: Please **DO NOT** rename the model files because their names are recorded in model information. Model files can be moved to another directory as long as `--modelDir` is specified in `infer`.

- **Parameters**
  - `--method`: [*required*] method used for the background model. Must be GLM or GBM.
  - `--featImpCut`: [*optional*] cutoff for feature importance score. Features with score >= cutoff will be used in the model. Only used for GLM. Default is 0.5.
  - `--gbmParam`: [*optional*] path to the parameter pickle for GBM. The pickle file must contain a valid python dictionary for XGBoost parameters.
  - `--gbmFold`: [*optional*] Number of model fold to train for GBM. Fold must be an integer >= 2. Default value is 3.
  - `--name`: [*optional*] Prefix for output files. Default is 'DriverPower'.
  - `--modelDir`: [*optional*] Directory for output model and model information files. Default is './output/'.
- **Notes**
  - Input data requirements can be found at *data format*.

- `--gbmFold` controls the split of training data by k-fold cross-validation. For example, default 3-fold model means the training data are divided into 3 equal folds. Each time, two folds of data are used to train the model and the rest fold is used to validate the model. Hence, three model files will be generated at the end. The prediction will then be the average of three models.
- Training phase can take hours for large training set and consume a large amount of memories. For example, using our default training set (~1M elements and 1373 features), training randomized lasso plus GLM with single core takes about 2-10 hours and 80G RAMs. Training 3-fold GBM with 15 cores takes about 10-24 hours and 80G RAMs.
- The default pickle file for `--gbmParam` is generated as follow:

```
import pickle

# Default parameter for XGBoost used in DriverPower
param = {'max_depth': 8,
         'eta': 0.05,  # learning rate
         'subsample': 0.6,
         'nthread': 15,  # number of threads; recommend using the number of available␣
↪CPUs
         'objective': 'count:poisson',
         'max_delta_step': 1.2,
         'eval_metric': 'poisson-nloglik',
         'silent': 1,
         'verbose_eval': 100,  # print evaluation every 100 rounds
         'early_stopping_rounds': 5,
         'num_boost_round': 5000  # max number of rounds; usually stop within 1500␣
↪rounds
        }

# Dump to pickle
with open('xgb_param.pkl', 'wb') as f:
    pickle.dump(param, f)
```

3.2 The `infer` sub-command

The `infer` sub-command is used to call drivers from test data with pre-trained models. Command-line options for this command can be viewed as follows:

```
$ driverpower infer -h
usage: driverpower infer [-h] --feature str --response str --modelInfo str
                         [--funcScore str]
                         [--method {auto,binomial,negative_binomial}]
                         [--scale float] [--funcScoreCut str] [--geoMean bool]
                         [--modelDir str] [--name str] [--outDir str]

optional arguments:
  -h, --help            show this help message and exit

input data:
  --feature str         Path to the test feature table (default: None)
  --response str        Path to the test response table (default: None)
  --modelInfo str       Path to the model information (default: None)
  --funcScore str       Path to the functional score table [optional]
                        (default: None)
```

(continues on next page)

(continued from previous page)

```
parameters:
  --method {auto,binomial,negative_binomial}
                        Test method to use [optional] (default: auto)
  --scale float         Scaling factor for theta in negative binomial
                        distribution [optional] (default: 1)
  --funcScoreCut str    Score name:cutoff pairs for all scores
                        e.g.,"CADD:0.01;DANN:0.03;EIGEN:0.3" [optional]
                        (default: None)
  --geoMean bool        Use geometric mean in test [optional] (default: True)
  --modelDir str        Directory of the trained model(s) [optional] (default:
                        None)
  --name str            Identifier for output files [optional] (default:
                        DriverPower)
  --outDir str          Directory of output files [optional] (default:
                        ./output/)
```

- **Input**
  - Required data: `--feature` and `--response` for test elements; `--modelInfo` from `driverpower model`.
  - Optional data: `--funcScore`. Only required for test with functional adjustment.
- **Output**
  - Driver discovery result saved in "`[outDir]/[name].result.tsv`".
- **Parameters**
  - `--method`: [*optional*] probability distribution used to generate p-values (binomial, negative_binomial or auto). Default is auto, which means decision will be made automatically based on the dispersion test of training data.
  - `--scale`: [*optional*] scaling factor of theta for negative binomial distribution. The theta is calculated from dispersion test. Default is 1. Only used for negative binomial distribution.
  - `--funcScoreCut`: [*optional*] Cutoff of each functional impact scores. The format of this parameter is a string in "NAME1:CUTOFF1;NAME2:CUTOFF2…", such as "CADD:0.01;DANN:0.03;EIGEN:0.3". Cutoff must in (0,1] and the name must match column names of `--funcScore`.
  - `--geoMean`: [*optional*] Whether to use geometric mean of nMut and nSample in test. Default is `True`.
  - `--modelDir`: [*optional*] Directory of model files from `driverpower model`. Only required when models have been moved to a different directory.
  - `--name`: [*optional*] Prefix for output files. Default is 'DriverPower'.
  - `--outDir`: [*optional*] Directory for output files. Default is './output/'.
- **Notes**

CHAPTER 4

Data format

4.1 Response table (`--response`)

The response (y; dependent variable) table records the observed number of mutations, number of mutated samples and the length per genomic element. This table is required for both `model` (training) and `infer` (test) sub-commands.

1. **Format**: TSV (tab-delimited text) with header. Compressed tables are also accepted.
2. **Fields**:
    - `binID`: identifier of genomic element and used as key.
    - `length`: effective length of the element.
    - `nMut`: number of observed mutations.
    - `nSample`: number of observed samples with mutations.
    - `N`: total number of samples.
3. **Example**:

binID	length	nMut	nSample	N
TP53.CDS				
KRAS.CDS				
...	...	...	...	...

4.2 Feature table (`--feature`)

The feature (X; independent variable) table records genomic features per element. This table can be large (~10GB) for thousands of features and ~1M genomic elements. This table is required for both `model` (training) and `infer` (test) sub-commands.

1. **Format**:

- TSV (or compressed TSV) with header. Loading may be slow for large datasets.
- HDF5 (*.h5 or *.hdf5). The HDF5 must contain key `X`, which is the feature table in pandas.DataFrame. Used for fast loading.

2. **Fields**:
   - `binID`: identifier of genomic element and used as key.
   - Column 2+: one feature per column. Unique feature names are required.
3. **Example**:

binID	GERP	E128-H3K27ac	...
TP53.CDS	4.80287	1.19475	...
KRAS.CDS	3.56563	2.53435	...

4.3 Feature importance table (`--featImp`)

The feature importance table records the feature importance score derived from randomized lasso, gradient boosting machine or other algorithms. This table can be used together with the `--featCut` argument to specify features kept in the background model. This table is only used and optional for the `model` sub-command with method `GLM`. If not provided, DriverPower will generate and output it automatically.

1. **Format**: TSV (or compressed TSV) with header.
2. **Fields**:
   - `name`: name of features, should match the column name in feature table.
   - `importance`: feature importance score.
3. **Example**:

name	importance
GERP	0.3
E128-H3K27ac	0.5
...	...

4.4 Functional score table (`--funcScore`)

The functional score table records the functional impact score per genomic element. DriverPower can work with any functional impact score scheme. This table is only used and optional for the `infer` sub-command.

1. **Format**: TSV (compressed TSV) with header.
2. **Fields**:
   - `binID`: identifier of genomic element and used as key.
   - Column 2+: one type of functional impact score per column.
3. **Example**:

binID	CADD	EIGEN	...
TP53.CDS	4.80287	1.19475	...
KRAS.CDS	3.56563	2.53435	...

4.5 Model and model information files (output of `model`)

4.6 Result table (output of `infer`)

CHAPTER 5

License

DriverPower is distributed under the terms of the GNU General Public License v3.0