User manual for *Trendsetter*

Mehreen R. Mughal and Michael DeGiorgio

August 31, 2018

Contents

1 Introduction

The *Trendsetter* software package can be used to calculate summary statistics $\widehat{\pi}$, r^2, N_{haps}, H_1, H_{12}, and H_2/H_1 in the same manner as in Mughal and DeGiorgio (2018), with these summary statistics used as input to train a trend-filtered multinomial regression model to differentiate among three classes: neutral, hard sweep, and soft sweep. The user can, however, provide their own set of summary statistics computed from an arbitrary number of K, $K \geq 2$, classes. Operation of this package requires a UNIX environment with Python 2.7 and R.3.4.1

Please cite this software as

MR Mughal, M DeGiorgio (2018) Localizing and classifying adaptive targets with trend filtered regression. *bioRxiv* doi:10.1101/320523

If you experience any issues, please contact Mehreen Mughal at mrm79@psu.edu for further help.

2 Operation

2.1 Installation

To download *Trendsetter*, visit

```
http://www.personal.psu.edu/mxd60/trendsetter.html
```

This download will include the user manual, software, and example datasets. The scripts included are designed to perform on a UNIX system. To unpack `trendsetter` from the command line, go to the directory where it is stored, and enter

```
tar -xzvf trendsetter.tar.gz
cd trendsetter/
```

The first command will decompress the file and release the content into folder `trendsetter/` in the current directory. The second command will shift the current directory to the `trendsetter/` directory, which contains the manual, `exampledata` directory, and the Python script `trendsetter.py`.

There are a number of required packages for this software to run correctly. The `fusedlasso` (Huling, 2014) package included with the software will be installed when the user first runs `trendsetter.py`. However, before running the software, the user will need to install the R packages `foreach` and `doParallel`, which enable quicker cross validation when training the classifier, as well as the Python package `NumPy`.

2.2 Calculating summary statistics from *ms*-formatted input files

Files in *ms* (Hudson, 2002) format typically contain multiple blocks of sequence data for a set of samples at segregating sites, where each block is typically a simulated replicate when based on simulated data. To calculate summary statistics from *ms*-formatted input files, use the command

```
python trendsetter.py calcstat <input_file>
```

This command will output a file `<input_file>.stats` in the same directory that `<input_file>` is located. The output file is CSV-formatted, and contains for each sequence block (*e.g.*, simulated replicate) a line with 1207 comma-separated columns. The first column of this output file is the position of the central SNP (test SNP) for a particular sequence block (*e.g.*, simulated replicate) from the *ms*-formatted input file. The next 1206 columns represent the six summary statistics ($\widehat{\pi}$, r^2, N_{haps}, H_1, H_{12}, and H_2/H_1) computed at

201 SNP data points ($6 \times 201 = 1206$) used as default by *Trendsetter*, as in Mughal and DeGiorgio (2018).

Specifically, for each line, $\widehat{\pi}$ has values located in columns $2, 8, 14, \ldots, 1202$, r^2 has values located in columns $3, 9, 15, \ldots, 1203$, N_{haps} has values located in columns $4, 10, 16, \ldots, 1204$, H_1 has values located in columns $5, 11, 17, \ldots, 1205$, H_{12} has values located in columns $6, 12, 18, \ldots, 1206$, and H_2/H_1 has values located in columns $7, 13, 19, \ldots, 1207$.

For *ms*-formatted input files, a single set of six summary statistics across 201 SNP data points is computed at the center of each block of sequences (*e.g.*, simulated replicate). Due to the manner in which summary statistics are computed in windows, the *ms*-formatted input file should contain at least 1010 segregating sites in each sequence block (*e.g.*, simulated replicate). If a sequence block (*e.g.*, simulated replicate) does not have at least 1010 segregating sites, then summary statistics will not be calculated for that block.

2.3 Calculating summary statistics from VCF files

To calculate summary statistics from data in VCF format (Danecek et al., 2011), use the command

```
python trendsetter.py calcstat_emp <input_file>
```

Summary statistics will be computed to enable classification of every fifth SNP beginning at the 505th SNP within the population data supplied. As described in Mughal and DeGiorgio (2018) and in section 2.2 above, each line of output contains a set of six summary statistics calculated across 201 SNP data points as well as the position of the test SNP. A number of temporary files will be created in the directory containing the VCF-formatted input file from which summary statistics are being generated. These files will be erased once the final output is created. The output summary statistics file will be named `<input_file>.stats` and will have the same format as the set of statistics calculated from *ms*-formatted files described in section 2.2, except that the first column of each output line will be the location of each test SNP in the VCF-formatted input file.

2.4 Training

In general, to train *Trendsetter* with K classes, use the command

```
python trendsetter.py train <classifier_name> <trend_penalty> <num_stats>
<class1.stats> <class2.stats> ...  <classK.stats>
```

where `<class1.stats>`, `<class2.stats>`, and `<classK.stats>` are summary statistic files for classes 1, 2, and K, respectively, `<classifier_name>` is a user-defined name for the classifier, `<trend_penalty>` is the trend filter penalty (either `constant` or `linear`) the user chooses to employ, and `<num_stats>` is the number of summary statistics m for which the spatial distribution is modeled in each class.

If the user chooses to use the default $m = 6$ summary statistics ($\widehat{\pi}$, r^2, N_{haps}, H_1, H_{12}, and H_2/H_1) calculated for $K = 3$ classes (*e.g.*, neutral, hard sweep, and soft sweep), then to train *Trendsetter* use the command

```
python trendsetter.py train <classifier_name> <trend_penalty> 6
<class1.stats> <class2.stats> <class3.stats>
```

where `<class1.stats>`, `<class2.stats>`, and `<class3.stats>` are summary statistic files that were output from the `calcstat` function (section 2.2), `<classifier_name>` is a user-defined name for the classifier, and `<trend_penalty>` is the trend filter penalty (either `constant` or `linear`) the user chooses to employ.

This `train` function will fit a multinomial logistic regression model with two penalties, a trend penalty, and a lasso penalty, by performing 10-fold cross validation across a grid of penalty values. After training, this

function will output three files: <classifier_name>_beta.csv containing the values of the regression coefficients, <classifier_name>_intercept.csv containing the intercept values, and <classifier_name>.std containing the means and standard deviations from the training files. These three files will be used to make predictions on test datasets. Prior to fitting the model, this function will standardize the input files. All three output files (<classifier_name>_beta.csv, <classifier_name>_intercept.csv, and <classifier_name>.std) will be necessary to apply *Trendsetter* to test datasets.

The user may also supply their own calculated summary statistics. Suppose the user computes m summary statistics at $2D + 1$ SNP data points (see Mughal and DeGiorgio, 2018). Then each line will contain $m(2D + 1) + 1$ comma-separated columns, where the first column represents the position of the test SNP, and the next $m(2D + 1)$ columns represent the summary statistics computed around the given test SNP. Furthermore, for each line, summary statistic s, $s = 1, 2, \ldots, m$, is located in columns $s+1, m+s+1, 2m+s+1, \ldots, 2Dm+s+1$. If the user is providing their own summary statistics, then all statistics must be calculated for the same number of SNP data points for each training and test dataset, and training and test sets must be identically formatted. We currently require that the user ensures a balanced training set, though we will perform the balancing in future updates to the software. Therefore, all training classes must contain the same number of training examples.

The user may also be interested in calibrating the probabilities output from *Trendsetter*. To calibrate the probabilities of the classifier, it is possible to train an additional multinomial logistic regression model using the probabilities output from *Trendsetter* as the independent variable and the true class as the dependent variable (Naeini, 2017). We recommend users to first visualize *Trendsetter* output through reliability curves to determine whether calibration is necessary, and visualize the output afterward to determine whether scaling has improved the calibration. To train a multinomial logistic regression model that will calibrate the probabilities of *Trendsetter*, use the command

```
python trendsetter.py cal <classifier_name> <class1.stats> <class2.stats>
<class3.stats>
```

It is important to note that the order of training data for the calibration must be the same as the order in the original classifier. The training data for this model should be separate from the *Trendsetter* training data and any testing datasets. The above command will output a file titled <classifier_namecalmodel.rds>

2.5 Testing

To apply *Trendsetter* to a test dataset, use the command

```
python trendsetter.py test <test_file> <classifier_name> <testout>
```

where <test_file> is a summary statistic file, <classifier_name> is the name of the classifier assigned in section 2.4, and <testout> is the name of the file where classifications for each test site will be stored. The <test_file> can be computed from *ms*-formatted data by using the `calcstat` command (section 2.2) from VCF-formatted data by using the `calcstat_emp` command (section 2.3), or can instead be computed by the user using the format specified in section 2.4 containing $m(2D + 1) + 1$ columns for m summary statistics computed at $2D+1$ SNP data points, with the first column representing the position of a test SNP.

The `test` function will standardize the summary statistics in <test_file> and output a file (<testout>) containing the probabilities of each class for each test site (line of <test_file>) using the classifier specified by input <classifier_name>.

To calibrate probabilities output from the test function above for a classifier for which you have already trained a calibration classifier using the command <cal> described in *Training* above, use the command

```
python trendsetter.py testcal <test_file> <classifier_name>
```

This will output a file `<test_file.calprobs>` containing calibrated probabilities from the input `<test_file>`.

3 Examples

We provide example *ms*-formatted simulated training data generated by `discoal` (Schrider and Kern, 2016) of hard sweep, soft sweep, and neutral scenarios for users to test the functionality of *Trendsetter*. The files entitled `Hardtrain`, `Softtrain`, and `Neuttrain`, located in the sub-directory `exampledata/examplems/`, contain 1000 independent simulated replicates each of hard sweep, soft sweep, and neutral scenarios, respectively. We provide test datasets in the sub-directory `exampledata/examplems/` simulated under identical parameters entitled `Hardtest`, `Softtest`, and `Neuttest`. The training and test sets provided were simulated under a constant population size, with model parameters under neutrality and selection as detailed in Mughal and DeGiorgio (2018).

If the user is only interested in quickly testing the training functionality of *Trendsetter*, then we provide smaller datasets entitled `smallhard.stat`, `smallsoft.stat`, and `smallneut.stat` for hard sweep, soft sweep, and neutral scenarios, in which summary statistics have been pre-computed and are located in sub-directory `exampledata/examplestats/`. We also provide a small portion of human chromosome 22 from 100 haplotypes sampled from the African Yoruban (YRI) population of the 1000 Genomes Project (The 1000 Genomes Project Consortium, 2015) in VCF file format.

To calculate summary statistics on each of the sample simulated files, use the following commands:

```
python trendsetter.py calcstat exampledata/examplems/Hardtrain

python trendsetter.py calcstat exampledata/examplems/Hardtest

python trendsetter.py calcstat exampledata/examplems/Softtrain

python trendsetter.py calcstat exampledata/examplems/Softtest

python trendsetter.py calcstat exampledata/examplems/Neuttrain

python trendsetter.py calcstat exampledata/examplems/Neuttest
```

These commands will output six files containing the summary statistics calculated for each of the simulated files titled `exampledata/examplems/<input_file>.stats` and will be located in sub-directory `exampledata/examplems/`, which is the directory of `<input_file>`.

To calculate summary statistics from the example VCF-formatted file provided, use the command

```
python trendsetter.py calcstat_emp exampledata/examplevcf/chr22TEST.vcf
```

This command will output a file `chr22TEST.vcf.stats` located in sub-directory `exampledata/examplevcf/` containing the summary statistics calculated from the provided region of human chromosome 22 from 100 YRI haplotypes.

To train *Trendsetter* with a constant ($d = 1$) tend penalty to differentiate among neutral, hard sweep, and soft sweep scenarios using the provided summary statistic files from the smaller dataset, use the command

```
python trendsetter.py train classifier1 constant 6
exampledata/examplestats/smallhard.stats exampledata/examplestats/smallneut.stats
exampledata/examplestats/smallsoft.stats
```

This command will output a model fit with a constant ($d = 1$) trend penalty named `classifier1`. The output includes the files `classifier1_beta.csv`, `classifier1_intercept.csv`, and `classifier1.std`, which are located in the same directory as `trendsetter.py`. Specifically, `classifier1_beta.csv` will contain for each class the regression coefficient values for each of the 1206 summary statistics described as described in section 2.2, `classifier1_interscept.csv` will contain the intercept values for each class, and `classifier1.std` will contain, as mentioned above in section 2.4, the standard deviations and means for each of the 1206 statistics across all classes.

Using the previously-trained classifier `classifier1`, we can classify a test dataset containing summary statistics computed for hard sweep simulations using the command

```
python trendsetter.py test exampledata/examplems/Hardtest.stats classifier1
Hardresult
```

This command will output a result file `Hardresult` containing the predicted class and probabilities of the three classes used to train `classifier1` for each the 1000 simulated datasets. The output file `Hardresult` will be located in the same directory as `trendsetter.py`.

Again, using the previously-trained classifier `classifier1`, we can classify the small region provided for human chromosome 22 based on the set of summary statistics calculted at SNP test positions using the command

```
python trendsetter.py test exampledata/examplevcf/chr22TEST.vcf.stats
classifier1 chr22res
```

This command will output a result file `chr22res` in which the first column indicates the test site, followed by the predicted class, and the probabilities for each class. The output file `chr22res` will be located in the same directory as `trendsetter.py`.

We also provide regression coefficients, intercepts, and standardization files for models calculated using simulations of seven human populations (YRI, LWK, GIH, TSI, CEU, CHB, and JPT) under the demographic histories inferred by Terhorst et al. (2017). These classifiers have been trained assuming three classes: neutral, hard sweep, and soft sweep, using a linear ($d = 2$) trend penalty based on the selection parameters described in Mughal and DeGiorgio (2018). These classifiers can be accessed by entering the population name (*i.e.*, either `YRI`, `LWK`, `GIH`, `TSI`, `CEU`, `CHB`, or `JPT`) preceded by the path as the `<classifier_name>` when using the `test` function. When using these classifiers, users must ensure that the test data set contains the same number of samples as the 1000 Genomes Project sample set for each population.

Population	**Number of haplotypes**
JPT	208
CHB	206
CEU	198
TSI	214
GIH	206
LWK	198
YRI	216

As an example, the supplied population model for JPT can be used for testing by entering the command

```
python trendsetter.py test <test_file> population_model/JPT/JPT <test_out>
```

This command will use the model trained to differentiate among hard sweeps, soft sweeps and neutrally-evolving regions in populations with demographic changes as specified by Terhorst et al. (2017) for the Japanese (JPT) population. Replacing `JPT` with `CHB`, `CEU`, `TSI`, `GIH`, `LWK`, or `YRI` will instead use the specified population model trained with that model demographic history.

Lastly, we would like to point out that fitting models with a linear trend penalty could take hours or days depending on the size of the user's dataset and available computational power, but fitting models with a constant trend penalty will complete within a few hours when using a data set comprised of 1206 statistics for 1000 training samples from each of three classes. We have noticed little difference in the classification rates and probability outputs between models trained with a constant versus linear trend penalty, trained and tested on the same datasets (Mughal and DeGiorgio, 2018).

References

P. Danecek, A. Auton, G. Abecasis, C. A. Albers, E. Banks, M. A. DePristo, R. E. Handsaker, G. Lunter, G. T. Marth, S. T. Sherry, G. McVean, R. Durbin, and . G. P. A. Group. The variant call format and vcftools. *Bioinformatics*, 27(15):2156–2158, 2011.

R. R. Hudson. Generating samples under a wright-fisher neutral model of genetic variation. *Bioinformatics*, 18:337–338, 2002.

J. Huling. fusedlasso. `https://github.com/jaredhuling/fusedlasso`, 2014.

M. R. Mughal and M. DeGiorgio. Localizing and classifying selective sweeps with trend filtered regression. *bioRxiv*, X:X, 2018. doi: 10.1101/320523.

M. P. Naeini. *Obtaining accurate probabilities using classifier calibration.* PhD thesis, University of Pittsburgh, 2017.

D. R. Schrider and A. D. Kern. Discoal: flexible coalescent simulations with selection. *Bioinformatics*, 32: 3839–3841, 2016.

J. Terhorst, J. A. Kamm, and Y. S. Song. Robust and scalable inference of population history from hundreds of unphased whole-genomes. *Nat Genet*, 49:303–309, 2017.

The 1000 Genomes Project Consortium. A global reference for human genetic variation. *Nature*, 526:68–74, 2015.