pyseer Documentation

Release 1.3.10

John Lees and Marco Galardini

Nov 21, 2023

CONTENTS:

pyseer was first written a python reimplementation of seer, which was written in C++. pyseer uses linear models with fixed or mixed effects to estimate the effect of genetic variation in a bacterial population on a phenotype of interest, while accounting for potentially very strong confounding population structure. This allows for genome-wide association studies (GWAS) to be performed in clonal organisms such as bacteria and viruses.

The original version of seer used sequence elements (k-mers) to represent variation across the pan-genome. pyseer also allows variants stored in VCF files (e.g. SNPs and INDELs mapped against a reference genome) or Rtab files (e.g. from roary or piggy to be used too). There are also a greater range of association models available, and tools to help with processing the output.

Testing shows that results (p-values) should be the same as the original seer, with a runtime that is roughly twice as long as the optimised C++ code.

We have also extended pyseer to fit association models to the whole genome, which also allows the use of machine learning to predict traits in new samples.

CHAPTER

ONE

CITATIONS

If you find pyseer useful, please cite:

Lees, John A., Galardini, M., et al. pyseer: a comprehensive tool for microbial pangenome-wide association studies. Bioinformatics 34:4310–4312 (2018). doi:10.1093/bioinformatics/bty539.

If you use unitigs (through unitig-counter) please cite:

Jaillard M., Lima L. et al. A fast and agnostic method for bacterial genome-wide association studies: Bridging the gap between k-mers and genetic events. PLOS Genetics. 14, e1007758 (2018). doi:10.1371/journal.pgen.1007758.

The whole genome/predictive models:

Lees, John A., Mai, T. T., et al. Improved inference and prediction of bacterial genotype-phenotype associations using interpretable pangenome-spanning regressions. (2020) Preprint: https://doi.org/10.1101/852426

1.1 Installation

The easiest way to install pyseer and its dependencies is through conda:

```
conda install pyseer
```

If you need conda, download miniconda and add the necessary channels:

```
conda config --add channels defaults
conda config --add channels bioconda
conda config --add channels conda-forge
```

pyseer can also be installed through pip:

```
python -m pip install pyseer
```

If you want multithreading make sure that you are using a version 3 python interpreter:

```
python3 -m pip install pyseer
```

1.1.1 Prerequisites

These modules are installed through the pip command above, but if you have cloned the repository you will need to install the depdencies yourself.

We used the following versions, though higher should also work:

- python 3+ (3.5.3)
- `numpy` (1.13.3)
- `scipy` (1.0.0)
- `pandas` (0.21.0)
- `scikit-learn` (0.19.1)
- `statsmodels` (0.8.0)
- `pysam` (0.13)
- `glmnet_py` (commit `946b65c`)
- `matplotlib` (2.1.0) – for scree plots
- `DendroPy` (4.3.0) – for phylogeny distances
- `pybedtools` (0.7.10) – for annotating k-mers
- `bedtools` (2.27.0) – for annotating k-mers
- `bedops` (2.4.9) – for annotating k-mers

1.1.2 Test installation

Run unit tests:

```
pytest -v tests
```

Test functions and output:

```
cd tests/ && bash run_test.sh && cd ../
```

1.1.3 Other software

To count k-mers, you may find fsm-lite or the original seer package useful. These can easily be installed with conda, set up as above:

```
conda install fsm-lite
conda install seer
```

Packages to count unitigs and help with their interpretation are also available on bioconda:

```
conda install unitig-counter unitig-caller
```

1.2 Option reference

Usage:

```
usage: pyseer [-h] --phenotypes PHENOTYPES
              [--phenotype-column PHENOTYPE_COLUMN]
              (--kmers KMERS | --vcf VCF | --pres PRES) [--burden BURDEN]
              [--distances DISTANCES | --load-m LOAD_M]
              [--similarity SIMILARITY | --load-lmm LOAD_LMM]
              [--save-m SAVE_M] [--save-lmm SAVE_LMM]
              [--mds {classic,metric,non-metric}]
              [--max-dimensions MAX_DIMENSIONS] [--no-distances]
              [--continuous] [--lmm] [--wg {enet,rf,blup}] [--lineage]
              [--lineage-clusters LINEAGE_CLUSTERS]
              [--lineage-file LINEAGE_FILE] [--sequence-reweighting]
              [--save-vars SAVE_VARS] [--load-vars LOAD_VARS]
              [--save-model SAVE_MODEL] [--alpha ALPHA] [--n-folds N_FOLDS]
              [--min-af MIN_AF] [--max-af MAX_AF] [--max-missing MAX_MISSING]
              [--filter-pvalue FILTER_PVALUE] [--lrt-pvalue LRT_PVALUE]
              [--cor-filter COR_FILTER] [--covariates COVARIATES]
              [--use-covariates [USE_COVARIATES ...]] [--print-samples]
              [--print-filtered] [--output-patterns OUTPUT_PATTERNS]
              [--uncompressed] [--cpu CPU] [--block_size BLOCK_SIZE]
              [--version]

SEER (doi: 10.1038/ncomms12797), reimplemented in python

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

Phenotype:
  --phenotypes PHENOTYPES
                        Phenotypes file (whitespace separated)
  --phenotype-column PHENOTYPE_COLUMN
                        Phenotype file column to use [Default: last column]

Variants:
  --kmers KMERS         Kmers file
  --vcf VCF             VCF file. Will filter any non 'PASS' sites
  --pres PRES           Presence/absence .Rtab matrix as produced by roary and
                        piggy
  --burden BURDEN       VCF regions to group variants by for burden testing
                        (requires --vcf). Requires vcf to be indexed

Distances:
  --distances DISTANCES
                        Strains distance square matrix (fixed or lineage
                        effects)
  --load-m LOAD_M       Load an existing matrix decomposition
  --similarity SIMILARITY
                        Strains similarity square matrix (for --lmm)
  --load-lmm LOAD_LMM   Load an existing lmm cache
  --save-m SAVE_M       Prefix for saving matrix decomposition
```

(continues on next page)

(continued from previous page)

```
  --save-lmm SAVE_LMM   Prefix for saving LMM cache
  --mds {classic,metric,non-metric}
                        Type of multidimensional scaling [Default: classic]
  --max-dimensions MAX_DIMENSIONS
                        Maximum number of dimensions to consider after MDS
                        [Default: 10]
  --no-distances        Allow run without a distance matrix

Association options:
  --continuous          Force continuous phenotype [Default: binary auto-
                        detect]
  --lmm                 Use random instead of fixed effects to correct for
                        population structure. Requires a similarity matrix
  --wg {enet,rf,blup}   Use a whole genome model for association and
                        prediction. Population structure correction is
                        implicit.
  --lineage             Report lineage effects
  --lineage-clusters LINEAGE_CLUSTERS
                        Custom clusters to use as lineages [Default: MDS
                        components]
  --lineage-file LINEAGE_FILE
                        File to write lineage association to [Default:
                        lineage_effects.txt]

Whole genome options:
  --sequence-reweighting
                        Use --lineage-clusters to downweight sequences.
  --save-vars SAVE_VARS
                        Prefix for saving variants
  --load-vars LOAD_VARS
                        Prefix for loading variants
  --save-model SAVE_MODEL
                        Prefix for saving model
  --alpha ALPHA         Set the mixing between l1 and l2 penalties [Default:
                        0.0069]
  --n-folds N_FOLDS     Number of folds cross-validation to perform [Default:
                        10]

Filtering options:
  --min-af MIN_AF       Minimum AF [Default: 0.01]
  --max-af MAX_AF       Maximum AF [Default: 0.99]
  --max-missing MAX_MISSING
                        Maximum missing (vcf/Rtab) [Default: 0.05]
  --filter-pvalue FILTER_PVALUE
                        Prefiltering t-test pvalue threshold [Default: 1]
  --lrt-pvalue LRT_PVALUE
                        Likelihood ratio test pvalue threshold [Default: 1]
  --cor-filter COR_FILTER
                        Correlation filter for elastic net (phenotype/variant
                        correlation quantile at which to start keeping
                        variants) [Default: 0.25]
```

(continues on next page)

(continued from previous page)

```
Covariates:
  --covariates COVARIATES
                        User-defined covariates file (tab-delimited, with
                        header, first column contains sample names)
  --use-covariates [USE_COVARIATES ...]
                        Covariates to use. Format is "2 3q 4" (q for
                        quantitative) [Default: load covariates but don't use
                        them]

Other:
  --print-samples       Print sample lists [Default: hide samples]
  --print-filtered      Print filtered variants (i.e. fitting errors) (does
                        not apply if --wg is used) [Default: hide them]
  --output-patterns OUTPUT_PATTERNS
                        File to print patterns to, useful for finding pvalue
                        threshold (not used with --wg)
  --uncompressed        Uncompressed kmers file [Default: gzipped]
  --cpu CPU             Processes [Default: 1]
  --block_size BLOCK_SIZE
                        Number of variants per core [Default: 3000]
  --version             show program's version number and exit
```

1.3 Best practices

1.3.1 Doing a genome-wide association study?

If you want to evaluate the effect of individual genomic variants on a phenotype of interest, while accounting for possible confounders, you will want to run a genome-wide association study. This will give a p-value for every genomic variant, comparing the alternative hypothesis that the variant does have an effect on phenotype (has an effect size $\beta > 0$) with the null hypothesis that the variant has no effect.

For this mode you will need at least three input files:

- Genetic variants (`--kmers`, `--vcf` or `--pres`).
- A phenotype (`--phenotypes`).
- A representation of the population structure (`--distances` or `--similarity`).

For a starting point, have a look at *GWAS tutorial*.

Current 'best-practice' GWAS recommendations:

- Use the `--lmm` mode.
- Use a phylogeny to generate the `--similarity` matrix.
- Use unitigs as the input, provided with the `--kmers` option. End-to-end analysis is identical to k-mers.
- If you have covariates, provide them with *–covariates* and *–use-covariates*.

Once this works, you may also wish to also add the following extra analyses:

- A burden test with `--vcf` and `--burden`.
- Tests of other forms of variation (genes, structural variants from panaroo.

- Extract lineage effects with `--lineage`.

1.3.2 Trying to predict a phenotype from genetics?

If you want to predict a phenotype in new samples where it is unmeasured, or look at the power of genetic variants to predict a phenotype, you'll want to use a whole-genome model.

You will need:

- Genetic variants (`--kmers`, `--vcf` or `--pres`).
- A phenotype (`--phenotypes`).

A good starting place is to read *Prediction tutorial*.

Current 'best-practice' prediction recommendations:

- Use `--wg enet --save-vars` and `--wg enet --load-vars` to save time in future runs.
- Use unitigs, if you can.
- For large variant sets, use a small number of `--cpu` to keep memory use manageable.
- Divide the population into strains with PopPUNK and use these definitions with `--lineage-clusters` and `--sequence-reweighting`.
- Turn the correlation filter off with `--cor-filter 0`.

1.3.3 Trying to calculate heritability?

If you want an estimate of what proportion of the phenotype variance can be explained by genomic variation, known as the heritability h^2, you can use either of the above modes to do this.

With `--lmm` an estimate for h^2 will be printed to stderr, based on the GCTA model (all variants affect the phenotype, with normally distributed effect sizes).

With `--wg enet` and estimate for h^2 will also be printed to stderr, based on the average prediction accuracy R^2 in held-out samples during cross-validation.

For a comparison of these approaches, see:

Lees, John A., Mai, T. T., et al. Improved inference and prediction of bacterial genotype-phenotype associations using interpretable pangenome-spanning regressions. (2020)

Preprint: https://doi.org/10.1101/852426

1.4 Usage

Quick start:

```
pyseer --phenotypes phenotypes.tsv --kmers kmers.gz --distances structure.tsv --min-af 0.
↪01 --max-af 0.99 --cpu 15 --filter-pvalue 1E-8 > pyseer.assoc
```

Will run the original `seer` model on given phenotypes and k-mers, using MDS scaling of the pairwise distances provided to correct for population structure. This will paralellize the analysis over 15 cores.

See the *Best practices* page for guidance on which options to use.

1.4.1 Input

pyseer will automatically take the intersection of samples found in the phenotype file and the population structure file. Only variation within these samples will be considered. Information on this is printed to STDERR.

Phenotype and covariates

The phenotype file is required to be supplied using the --phenotypes option. The format is tab-delimited, with the sample name in the first column, and the phenotype in the last column. A header is required as the first row:

```
samples continuous      binary
sample_1        1       0
sample_2        2       1
sample_3        3       1
sample_4        4       1
sample_5        5       1
sample_6        6       1
sample_7        7       0
```

The default column to use as the phenotype is the last column, but you can provide an explicit value with --phenotype-column. Missing phenotypes can be supplied as 'NA'. If all values are 0 or 1 a binary phenotype is assumed (only relevant for the fixed effect model), otherwise a continuous phenotype is used. Use --continuous to force this behaviour.

Warning: Using numbers as the sample names has been reported to cause problems in some modes and versions of pyseer. While we have tried to fix this issue, if you run into trouble try chaning your sample names into a string (e.g. by adding an underscore at the end of every name).

Covariate files (--covariates) must be tab-delimited with a header row, and the first column must contain the sample names:

```
samples      time       cluster
sample_1        1       cluster1
sample_2        2       cluster2
sample_3        3       cluster0
sample_4        4       cluster1
sample_5        5       cluster2
sample_6        6       cluster0
sample_7        7       cluster1
```

Choose which covariates to use with --use-covariates. Provide space separated column numbers to use. The default is that the covariates are labels, but for a quantitative covariate add 'q' after the column number. For the above example --use-covariates 2q 3 would be the correct argument.

k-mers

Variable length k-mers counted by fsm-lite or dsm-framework are input with the `--kmers` option. This file is assumed to be gzipped, use the `--uncompressed` option if they aren't. If you wish to use dsk to count k-mers you will need to use `combineKmers` from the original `seer` installation to convert them to the correct input format.

If needed, both fsm-lite and seer can be installed through conda. See *Installation* for details.

Note: For common variation k-mers or unitigs should probably be your variant of choice. `seer` was mainly designed to work with k-mers, due to their ability to test variation across the pan-genome without the need to call variants against multiple references, or deal with the complexities of constructing accurate COGs for the whole population. We have included these input formats for convenience and flexibility.

We would recommend the use of SNPs and genes *in addition* to k-mers, or for a quick first pass analysis.

unitigs

Unitigs are nodes in a compressed de Bruijn graph, and remove some of the redundancy present in k-mer counting, as well as presenting fewer tests (and advantage both computationally and statistically) and being easier to interpret thanks to their length and context provided by the variation graph.

Unitigs can both be counted, and called consistently in new populations, using the unitig-caller package.

An older version of the package, giving the same results, is available as unitig-counter (see documentation in the `README.md`).

Usage is then identical to k-mers, providing input with the `--kmers` options, and `--uncompressed` if necessary.

Note: Both packages can be installed thorough conda, see *Installation* for details.

SNPs and INDELs

Short variation (SNPs and INDELs) can be read from a VCF file using the `PySAM` module. Simply use the `--vcf` option to read in your file.

If you have multiple VCF files (e.g. one per sample) you can combine them with `bcftools`:

```
bcftools merge -m none -0 -O z *.vcf.gz > merged.vcf.gz
```

Sample names are taken from the header row. Only one `ALT` variant per row is supported, if you have multiple alternative variants use:

```
bcftools norm -m - <in.vcf> > out.vcf
```

to split them into multiple rows otherwise they will be skipped. If `FILTER` fields are present only those with 'PASS' will be processed.

Note: The `GT` field is used to determine variant presence/absence. '0' or '.' is absence, anything else is presence.

Genes and intergenic regions, or any other variant type

COG or intergenic region variation is represented as an .Rtab file by roary and piggy:

```
Gene sample_1        sample_2
COG1 1       1
COG2 1       0
```

These can be used directly with `--pres`, and this format can be used flexibly to represent variants from other sources.

Rare variants

`pyseer` supports burden testing of rare variants. Variants at low frequency which are associated with the phenotype cannot be detected by a standard regression model. A burden test groups sets of rare variants with the same predicted biological effect, and then treats these sets like common variants.

Note: Group variants only with the same predicted functional effect. A good start would be all loss of function mutations (frameshift or stop gained/nonsense) within a gene. This can be expanded to operons or pathways, and to variants predicted as damaging (missense) or all variants. Burden tests assume all variants in a group have the same direction of effect, and will lose power if this assumption is broken.

To run a burden test, available under any of the association models below, requires a VCF file of SNPs and INDELs. First predict the function of mutations (using VEP or bcftools csq) and filter the VCF file appropriately on variant frequency and predicted effect:

```
bcftools view -Q 0.01 -i 'CSQ[*] ~ "stop_gained" snps_indels.vcf.gz | CSQ[*] ~
↪"frameshift_variant"' | bgzip -c > low_freq_vars.vcf.gz
```

Then run `pyseer` providing a list of regions to group variants by to the `--burden` option and the filtered VCF file with `--vcf`. These regions are one per line, with their name and the bcftools style region co-ordinates:

```
CDS1    FM211187:3910-3951
CDS2    FM211187:4006-4057
```

Multiple regions can be specified for a single burden test, by separating each region using a comma:

```
pathway1    FM211187:4006-4057,FM211187:5673-5777
```

Warning: The same frequency filters as for common variants still apply. Only groups within the threshold will be tested. To ensure only rare variants enter the sets, you will need to pre-filter the VCF file with bcftools as shown above.

Filtering

Filtering on allele frequency is necessary, unless the input has already been filtered. We would recommend only including variants with a minor allele count of at least five. Use `--min-af` and `--max-af` to achieve this. The default is to test variants with a MAF > 1%.

If computational resources are limited, you can use the unadjusted p-value as a pre-filter `--filter-pvalue`. 10^{-5} is a reasonable value, or three orders of magnitude below your final significance threshold. If you just want to plot the significant results, or save space in the output you can also print just those passing a final threshold with `--lrt-pvalue`.

> **Warning:** We would recommend not filtering on p-value if possible. It is possible that variants not significant before correction may be significant afterwards, and taking a final threshold will prevent a Q-Q plot from being used to test for inflation of p-values.

1.4.2 Population structure

To adjust for population structure, the fixed effects (*Fixed effects (SEER)*) model needs a matrix with distances between all pairs of samples in the analysis:

```
     sample_1        sample_2        sample_3
sample_1     0       0.0115761       0.0119383
sample_2     0.0115761       0.0     0.0101878
sample_3     0.0119383       0.0101878       0.0
```

This file is included with `--distances`. The default is to perform classical MDS on this matrix and retain 10 dimensions. The type of MDS performed can be changed with the `--mds` option to metric or non-metric if desired. Once the MDS has run once, the `--save-m` argument can be used to save the result to file. Subsequent runs can then be provided with this decomposition directly using `load-m` rather than recomputing the MDS.

An alternative to using a distance matrix in the fixed effects analysis is to provide clusters of samples with the same genetic background (e.g. from BAPS) as a categorical covariate with the `--use-covariates` option. In this case you should also add the `--no-distances` options to allow running without one of the matrices below, which would define these covariates twice.

The mixed effects model (*Mixed model (FaST-LMM)*) needs a matrix with covariances/similarities included with `--similarities` between all pairs of samples in the analysis:

```
     sample_1        sample_2        sample_3
sample_1     0.319   0.004   0.153
sample_2     0.004   0.004   0.004
sample_3     0.153   0.004   0.288
```

This is known as the kinship matrix K. Analagously to the MDS runs, the decomposition can be save with `--save-lmm` and loaded with `--load-lmm` in subsequent analysis rather than processing the similarity matrix again.

Both types of matrix are necessarily symmetric. The entries along the diagonal of a pairwise distance matrix are zeros. The matrices can be generated in three ways.

mash

mash can be used to rapidly estimate distance between samples. First of all create a sketch of all your samples (assuming assembled contigs in fasta files):

```
mash sketch -s 10000 -o samples *.fa
```

Calculate the pairwise distances and create a distance matrix:

```
mash dist samples.msh samples.msh | square_mash > mash.tsv
```

These distances can only be used with the fixed effects model.

Phylogeny based

If you have a high quality phylogeny (removing recombination, using a more accurate model of evolution) using this to calculate pairwise distances may be more accurate than mash. For the fixed effects model you can extract the patristic distances between all samples. Using a newick file:

```
python scripts/phylogeny_distance.py core_genome.tree > phylogeny_distances.tsv
```

For use with *Mixed model (FaST-LMM)* add the `--calc-C` or `--lmm` option (which are equivalent). This calculates the similarities based on the shared branch length between each pair's MRCA and the root (as PDDIST):

```
python scripts/phylogeny_distance.py --lmm core_genome.tree > phylogeny_similarity.tsv
```

If you want to ignore branch lengths (not usually recommended) use the `--topology` option. Other tree formats supported by dendropy can be used by specifying `--format`.

Genotype matrix

For a mixed model association the FaST-LMM default is to use the genotype matrix (design matrix) of variant presence absence to calculate the kinship matrix $K = GG^T$. To use this method for the `--similarity` option use the similarity script with any valid pyseer input variant type:

```
similarity_pyseer --vcf core_gene_snps.vcf sample_list.txt > genotype_kinship.tsv
```

Where `sample_list.txt` is a file containing sample names to keep, one on each line.

> **Warning:** Choose the input to this command carefully. Using too few variants or those which don't represent vertical evolution may be inaccurate (e.g. the roary gene presence/absence list). Choosing too many will be prohibitive in terms of memory use and runtime (e.g. all k-mers). A VCF of SNPs from the core genome is a good tradeoff in many cases.

No population structure correction

You can run the fixed effects model without a population structure correction. As this is generally not recommended you need to add the `--no-distances` option to allow the analysis to run.

Situations where this may be desirable are when you are using population structure(/lineage) as the phenotype i.e. looking for k-mers which define lineages, or if you are correcting for population structure manually using covariates such as cluster IDs.

1.4.3 Association models

Symbols used:

Symbol	Meaning
y	A vector containing the phenotype for each sample.
W	A design matrix containing the covariates, and the MDS components if SEER's model is used.
a	Fixed effects for the covariates.
X	A design matrix (/vector) containing the variant presence/absence.
b	Fixed effects for the variant (also known as beta/effect size).
K	The kinship matrix of relations between all pairs of samples.
G	The genotype matrix of all variant presence/absence.
u	Random effects for each row of the kinship matrix.

Fixed effects (SEER)

If provided with a valid phenotype and variant file this is the default analysis run by `pyseer`. In summary, a generalized linear model is run on each k-mer (variant), amounting to multiple linear regression for continuous phenotypes and logistic regression for binary phenotypes. Firth regression is used in the latter case when large effect sizes are predicted. For details see the original publication.

$$y \sim Wa + Xb$$

The most important adjustment to this analysis is choosing the number of MDS components with the `--max-dimensions` argument. Once you have your `--distances` matrix, draw a scree plot:

```
scree_plot_pyseer mash.tsv
```

This will show the variance explained (the eigenvalues of each MDS component) for the first 30 dimensions (increased using `--max-dimensions` to `scree_plot_pyseer`). You can pick a value at the 'knee' of this plot, or choose to include much of the total variation. Consider choosing around the first 30 components.

Mixed model (FaST-LMM)

A linear mixed model (LMM) of fixed and random effects can be fitted by adding the `--lmm` option, as well as either `--similarities` or `--load-lmm` from a previous analysis.

$$y \sim Wa + Xb + Ku$$

We use FaST-LMM's likelihood calculation to compute this model in linear time for each variant. The phenotype is always treated as continuous, which in the case of case/control data may cause some loss of power.

The main advantage of this model is that all relationships are implicitly included and selection of the number of components to retain is not necessary. In comparison to the fixed effect model this has shown to better control inflation of p-values (https://elifesciences.org/articles/26255).

In addition this model will output the narrow sense heritability h^2, which is the proportion of variance in phenotype explained by the genetic variation when maximizing the log-likelihood:

$$LL(\sigma_E^2, \sigma_G^2, \beta) = \log N(y|X\beta; \sigma_G^2 K + \sigma_E^2 I)$$
$$h^2 = \frac{\sigma_G^2}{\sigma_G^2 + \sigma_E^2}$$

This assumes effect sizes are normally distributed, with a variance proportional to the total genetic variance (the GCTA model). See this paper for more information on the heritability of pathogen traits.

Warning: pyseer will print the h^2 estimate to STDERR, but it will only be valid under the assumptions of the model used. You may wish to compare estimates from other software, and particular care should be taken with binary phenotypes.

Whole genome models (elastic net)

All variants can be included at once with the `--wg` mode. Currently only the elastic net is implemented, but more models will be included in future.

An elastic net can be fitted to all the variants at once by providing the `--wg enet` option, using the glmnet package to solve the following problem:

$$\min_{b_0,b} \frac{1}{N} \sum_{i=1}^{N} w_i l(y_i, b_0 + b^T x_i)^2 + \lambda \left[(1-\alpha)||b||_2^2/2 + \alpha||b||_1 \right]$$

with the link function $w_i l()$ set by the phenotype error distribution.

In this mode, all the variants are read into an object in memory, a correlation-based filter is applied, the model is fitted, then those variants with non-zero b are printed in the output. The model is fit by ten-fold cross-validation to pick the λ which gives the lowest deviance when compared to the true phenotypes. Higher λ leads to smaller fitted b values. These values, along with the corresponding best R^2 will be written to `STDERR`. Setting α closer to one will remove more variants from the model by giving them zero beta.

Tip: Population structure can be included using `--sequence-reweighting` and `--lineage-clusters`. Use of the latter will also use these clusters to give a more representative cross-validation accuracy. See *Prediction tutorial* for more details.

Cross-validation uses `--cpu` threads, which is recommended for better performance.

Warning: As all variants are stored in memory, and potentially copied, very large variant files will cause this method to run out of RAM. We therefore do not recommend running on k-mers, but to use unitigs instead. SNPs and genes work fine.

By default, the top 75% of variants correlated with the phenotype are included in the fit. Variants will include the unadjusted single-variate p-values, if distances have been provided with either `--distances` or `--load-m` the adjusted p-values will also be present.

Option	Use
`--save-va`	Save the object representing all objects to disk. Useful for reruns, or using multiple phenotypes.
`--load-va`	Load the variants saved to disk, the most time-consuming step.
`--save-mc`	Save the fitted model so that one can perform *Prediction with the elastic net* on samples with unobserved phenotypes.
`--alpha`	Sets the mixing between ridge regression (0) and lasso regression (1) in the above formula. Default is 0.0069 (closer to ridge regression)
`--n-folds`	Number of folds in cross validation (samples removed to test prediction accuracy). Default is 10.
`--cor-fil`	Set the correlation filter to discard the variants with low correlation to the phenotype. Default is 0.25 (keeping the top 75% variants correlated with phenotype).

Note: When using `--load-vars` you still need to provide the original variant file with `--vcf`, `--kmers` or `--pres` as this is read again to output the selected variants. pyseer will test that the checksums of this files is identical to that used with `--save-vars`, and will warn if any difference is detected.

Prediction with the elastic net

If `--wg` was used with `--save-model` this fit can be used to attempt to predict the phenotype of new samples without a phenotype label:

```
enet_predict --vcf new_snps.vcf.gz old_snps.lasso_model.pkl samples.list > lasso.
↪predictions.txt
```

Provide the samples you wish to predict the phenotype of in `samples.list` along with comparable variants and covariates to that which were used in the original model. If any variant or covariate is not found in the new input this will be noted on `STDERR` and the mean values (the originally observed allele frequency) will be used instead. Use `--ignore-missing` to turn this off.

See *Prediction tutorial* for more examples.

Lineage effects (bugwas)

Earle et al introduced the distinction between 'lineage' and 'locus' effects. Also see this review. The p-values output by `pyseer` are aimed at finding 'locus' effects. To find lineage effects Earle et al proposed ordering variants by those associated with both the phenotype and a lineage highly associated with a phenotype. They performed this by decomposing the random effects to find the principal component each variant was most associated with, and then order variants by those principal components most associated with the phenotype.

To perform a similar analysis in `pyseer`, add the `--lineage` option. This first checks the lineages most associated with the phenotype:

$$y \sim Wa$$

writing the results to `--lineage_file`, ordered by the most associated lineage. For each variant, after the main regression the lineage the variant belongs to is chosen by the most significant when regressing the variant presence/absence on the lineages:

$$X \sim Wa$$

To pick lineage effects, those variants assigned to a lineage highly associated with the phenotype in the `--lineage_file` and with a significant p-value should be chosen. A Manhattan plot, with the x-axis order defined by the lineage column in the output, can be created.

The default is to use the MDS components to define lineage effects, but you can supply custom lineage definitions such as BAPS clusters with the `--lineage-clusters` options:

```
sample_1        BAPS_3
sample_2        BAPS_16
sample_3        BAPS_27
sample_4        BAPS_3
```

Note: One of these clusters will be removed to ensure the regressions are of full rank. Therefore there is one cluster variants will never be assigned to. This is chosen as the cluster least associated with the phenotype.

1.4.4 Output

`pyseer` writes output to STDOUT, which you can redirect with a pipe >. The format is tab separated, one line per variant tested and passing filtering, with the first line as a header. Add `--print-samples` to print the k-samples and nk-samples fields.

Fields for a fixed effect analysis:

Field	Meaning
variant	sequence of k-mer or ID of variant from VCF or Rtab.
af	allele frequency. The proportion of samples the variant is present in.
filter-pvalue	association of the variant with the phenotype, unadjusted for population structure.
lrt-pvalue	the p-value of association, adjusted for population structure. This corresponds to the LRT p-value of `seer`.
beta	the effect size/slope of the variant. For a binary phenotype, exponentiate to obtain the odds-ratio.
beta-std-err	the standard error of the fit on beta.
intercept	the intercept of the regression.
PCX	the slope each fixed effect (covariate and MDS component).
k-samples (optional)	the samples the variant is present in (comma separated).
nk-samples (optional)	the samples the variant is not present in (comma separated).
lineage (optional)	the lineage the variant is most associated with.
notes	notes about the fit.

Fields for a mixed model analysis:

Field	Meaning
variant	sequence of k-mer or ID of variant from VCF or Rtab.
af	allele frequency. The proportion of samples the variant is present in.
filter-pvalue	association of the variant with the phenotype, unadjusted for population structure.
lrt-pvalue	the p-value from the mixed model association, as given by FaST-LMM.
beta	the effect size/slope of the variant.
beta-std-err	the standard error of the fit on beta.
variant_h2	the variance in phenotype explained by the variant. The h^2 for this variant alone.
k-samples (optional)	the samples the variant is present in
nk-samples (optional)	the samples the variant is not present in
lineage (optional)	the lineage the variant is most associated with.
notes	notes about the fit.

Notes field

Possible 'notes' are:

Note	Meaning
af-filter	Variant failed set allele frequency filters `--min-af` or `--max-af`.
pre-filtering-failed	Variant failed `filter-pvalue` filter .
lrt-filtering-failed	Variant failed `lrt-pvalue` filter.
bad-chisq	χ^2 test was invalid, suggesting either a very high effect size or low allele frequency. Firth regression used.
high-bse	SE of fit was >3, which may imply a high effect size. Firth regression used.
perfectly-separable-data	Variant presence and phenotype exactly correlate, so regression cannot be fitted.
firth-fail	Firth regression failed (did not converge after 1000 iterations).
matrix-inversion-error	A pseudo-inverse could not be taken, preventing model from being fitted. This likely implies nearly separable data.
missing-data-error	Model could not be fitted because of missing data or `inf` values.

Number of unique patterns

One way to pick the threshold for significance is to use a Bonferroni correction with the number of unique variant patterns as the number of multiple tests. When running `pyseer` add the `--output-patterns` option to write a file with hashes of the patterns.

Then run the `count_patterns.py` script on this output:

```
python scripts/count_patterns.py --alpha 0.05 --cores 4 --memory 1000 --temp /tmp patterns.txt
```

This will return the number of unique patterns and the significance threshold. `--alpha` is the unadjusted significance threshold to use. The other options interface to GNU `sort` to speed up the calculation, and control the amount of data stored in main memory/where to store on disk.

Effect sizes

The effect size is referred to as β. For a binary phenotype, fitted with the fixed effect model, the odds ratio can be calculated with e^{β}. For continuous phenotypes or if using the linear mixed model (even with a binary phenotype) the β roughly gives the absolute increase in probability.

For example, if a 5% of samples without a variant have a phenotype, and $\beta = 0.6$, then around 65% of samples with the variant would be expected to have the phenotype, giving an odds ratio of $\frac{0.65}{0.05} = 13$.

For a more accurate transformation, see this article and the accompanying shiny app.

1.4.5 Processing k-mer output

See the *GWAS tutorial* for full concrete examples.

Mapping to references (phandango)

K-mers can be mapped to reference genomes using the provided script and a fasta file of the reference:

```
phandango pyseer_kmers.assoc reference_1.fa reference_1.plot
```

These `.plot` files can be dragged and dropped into phandango along with a reference annotation file (the `.gff` file corresponding to the fasta reference file). Phandango will display the length of the k-mer as well as its position. The y-axis is $-\log_{10}(p)$.

> **Warning:** If all the k-mers are plotted performance will be slow. It is computationally challenging to render tens of millions of k-mers with a real time interface, so we recommend filtering out those with a p-value below a threshold value for interactive performance.

Annotating k-mers

K-mers can also be annotated with the gene they are in, or nearby. This requires a list of annotations. Trusted references are used first, and allow a close match of k-mer (using `bwa mem`). Draft annotations, ideally those the k-mers were counted from, are used second, and require an exact match of the k-mer (using `bwa fastmap`).

K-mers will be iteratively mapped to references in the order provided, either until all the references are used, or all k-mers have been mapped:

```
annotate_hits_pyseer pyseer_kmers.assoc references.txt kmer_annotation.txt
```

The `references.txt` file contains the sequence, annotation and type of the references to be used:

```
D39.fa       D39.gff ref
TIGR4.fa     TIGR4.gff       ref
sample1.fa   sample1.gff     draft
sample2.fa   sample2.gff     draft
```

To map all of the k-mers, and ensure good quality annotation where possible, provide a few trusted references as the first lines in this file. You can then list all of the assemblies used as input after this, designated as draft.

For each k-mer, each match will be returned in the format 'contig:pos;gene_down;gene_in;gene_up' i.e. the closest downstream gene, the gene the k-mer is in (if it is), the closest upstream gene. The gene name will be chosen if in the GFF, otherwise the gene ID will be used.

Note: This analysis uses bedtools to find overlapping and nearby genes. A working installation of bedtools is therefore required. The construction of each query is slow, so only significant k-mers should be annotated in this manner.

Note: By default `annotate_hits_pyseer` will only consider CDS features in the provided GFF files. If you want to consider other feature types you can use the `--feature-type` option (*e.g.* `--feature-type rRNA --feature-type tRNA`).

To summarise these annotations over all significant k-mers, use the `summarise_annotations.py` script:

```
python scripts/summarise_annotations.py kmer_annotation.txt
```

For each gene name, the number of overlapping significant k-mers, maximum p-value, average MAF and average effect size will be reported. This is ideal input for plotting with ggplot2.

1.4.6 Processing unitig output

As unitigs are sequence elements of variable length, identical steps can be taken as for k-mers, as described above.

Additionally, `cdbg-ops` provided by installing `unitig-counter` can be used to extend short unitigs leftwards and rightwards by following the neighbouring nodes in the de Bruijn graph. This can help map sequences which on their own are difficult to align in a specific manner.

Create a file `unitigs.txt` with the unitigs to extend (probably your significantly associated hits) and run:

```
cdbg-ops extend --graph output/graph --unitigs unitigs.txt > extended.txt
```

The output `extended.txt` will contain possible extensions, comma separated, with lines corresponding to unitigs in the input. See the help for more options.

1.5 GWAS tutorial

For a short introduction to bacterial GWAS, you may wish to read this review.

This tutorial shows how to use `pyseer` to perform a GWAS for penicillin resistance using 616 *S. pneumoniae* genomes collected from Massachusetts. These genomes were first reported here and can be accessed here. One of the earliest GWAS studies in bacteria was performed using this data, and we will try to replicate their results.

The data for this tutorial can be accessed here. Extract the archive:

```
tar xvf pyseer_tutorial.tar.bz2
```

To find the following files:

File	Contents
assemblies.tar.bz2	Archive of genome assemblies.
fsm_file_list.txt	Input to run fsm-lite.
snps.vcf.gz	SNPs mapped against the Spn23F reference.
gene_presence_absence.Rtab	Output from roary run on these genomes.
core_genome_aln.tree	IQ-TREE phylogeny (using `-m GTR`) from the core genome alignment.
resistances.pheno	Whether an isolate was resistant to penicillin, to be used as the phenotype.
mash_sketch.msh	mash sketch output, from running `mash sketch -s 10000 -o mash_sketch *.fa`.
Spn23F.fa	23FSpn sequence.
Spn23F.gff	23FSpn sequence and annotation.
6952_7#3.fa	The draft sequence assembly of one isolate in the collection.
6952_7#3.gff	The draft annotation of the isolate.

Note: To run commands with the scripts/ directory you will need to have cloned the github repository (though other commands can continue to be run using your conda/pip install.

1.5.1 SNP and COG association with fixed effects model

We will first of all demonstrate using `pyseer` with the original `seer` model, using MDS components as fixed effects to control for the population structure. We will test the association of SNPs mapped to a reference (provided as a VCF file) and COG presence/absence (provided as and Rtab file, from running roary on the annotations).

The first step is to estimate the population structure. We will do this using a pairwise distance matrix produced using `mash`. Either create the mash sketches yourself:

```
mkdir assemblies
cd assemblies
tar xf ../assemblies.tar.bz2
cd ..
mash sketch -s 10000 -o mash_sketch assemblies/*.fa
```

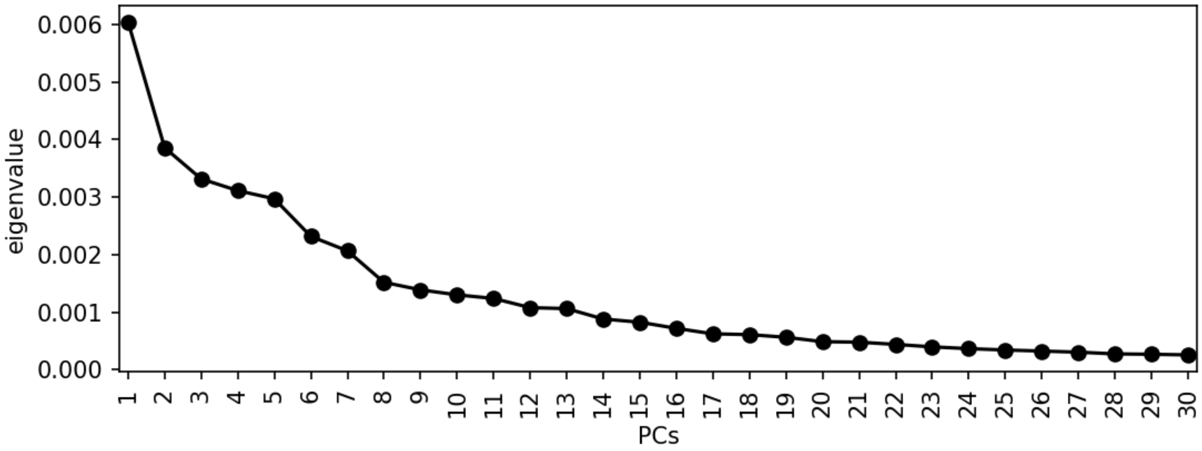
or use the pre-computed `mash_sketch.msh` directly. Next, use these to calculate distances between all pairs of samples:

```
mash dist mash_sketch.msh mash_sketch.msh| square_mash > mash.tsv
```

Note: Alternatively, we could extract patristic distances from a phylogeny: `python scripts/phylogeny_distance.py core_genome_aln.tree > phylogeny_dists.tsv`

Let's perform an MDS and these distances and look at a scree plot to choose the number of dimensions to retain:

```
scree_plot_pyseer mash.tsv
```



There is a drop after about 8 dimensions, so we will use this many. This is subjective, and you may choose to include many more. This is a sensitivity/specificity tradeoff – choosing more components is more likely to reduce false positives from population structure, at the expense of power. Using more components will also slightly increase computation time.

We can now run the analysis on the COGs:

```
pyseer --phenotypes resistances.pheno --pres gene_presence_absence.Rtab --distances mash.
↪tsv --save-m mash_mds --max-dimensions 8 > penicillin_COGs.txt
```

Which prints the following to STDERR:

```
Read 603 phenotypes
Detected binary phenotype
Structure matrix has dimension (616, 616)
Analysing 603 samples found in both phenotype and structure matrix
10944 loaded variants
4857 filtered variants
6087 tested variants
6087 printed variants
```

`pyseer` has automatically matched the sample labels between the inputs, and only used those which were present in the phenotype file. This has accounted for the fact that not all of the samples were measured for the current phenotype. We have used the default filters, so only intermediate frequency COGs have been considered. The core genome COGs and low frequency COGs are in the 4857 filtered out. Take a look at the top hits:

```
sort -g -k4,4 penicillin_COGs.txt | head

variant af      filter-pvalue   lrt-pvalue      beta    beta-std-err    intercept      ␣
↪PC1     PC2     PC3     PC4     PC5     PC6     PC7     PC8 notes
group_4276      7.79E-02        1.27E-11        2.70E-21        1.29E+01        7.12E-01␣
↪       -1.29E+00       -7.01E-01       -2.75E+00       -6.64E+00       -9.02E-01       1.
↪46E+01         -3.83E+00       -6.05E-01       -4.25E+00       high-bse
group_4417      8.96E-02        3.21E-09        4.72E-20        -6.08E+00       6.99E-01␣
↪       -4.51E-01       -1.12E+00       5.08E-01        -5.61E+00       8.20E-01        8.
↪19E+00         -4.95E-01       -4.53E-01       9.70E-01        bad-chisq
cpsG    1.18E-01        1.34E-16        1.69E-19        3.77E+00        5.25E-01        -
↪1.34E+00       2.49E+00        1.24E-01        -5.19E+00       6.57E-01 1.01E+01       8.
```

(continues on next page)

(continued from previous page)

```
↪38E-02        -3.06E-01       8.48E-01
group_3096      1.18E-01        1.34E-16        1.69E-19        3.77E+00        5.25E-01␣
↪        -1.34E+00       2.49E+00        1.24E-01        -5.19E+00    6.57E-01        1.
↪01E+01         8.38E-02        -3.06E-01       8.48E-01
group_5738      1.18E-01        1.34E-16        1.69E-19        3.77E+00        5.25E-01␣
↪        -1.34E+00       2.49E+00        1.24E-01        -5.19E+00    6.57E-01        1.
↪01E+01         8.38E-02        -3.06E-01       8.48E-01
group_8161      1.18E-01        1.34E-16        1.69E-19        3.77E+00        5.25E-01␣
↪        -1.34E+00       2.49E+00        1.24E-01        -5.19E+00    6.57E-01        1.
↪01E+01         8.38E-02        -3.06E-01       8.48E-01
group_8834      1.18E-01        1.34E-16        1.69E-19        3.77E+00        5.25E-01␣
↪        -1.34E+00       2.49E+00        1.24E-01        -5.19E+00    6.57E-01        1.
↪01E+01         8.38E-02        -3.06E-01       8.48E-01
mnaA    1.18E-01        1.34E-16        1.69E-19        3.77E+00        5.25E-01        -
↪1.34E+00       2.49E+00        1.24E-01        -5.19E+00    6.57E-01 1.01E+01        8.
↪38E-02        -3.06E-01       8.48E-01
tagA    1.18E-01        1.34E-16        1.69E-19        3.77E+00        5.25E-01        -
↪1.34E+00       2.49E+00        1.24E-01        -5.19E+00    6.57E-01 1.01E+01        8.
↪38E-02        -3.06E-01       8.48E-01
```

Note that the first two rows have notes `high-bse` and `bad-chisq` respectively. For the former this may represent a high effect size, low frequency results. For the latter this is likely due to the MAF filter not being stringent enough. The identical p-values of the other results are as these COGs appear in exactly the same set of samples.

We will now perform an analysis using the SNPs produced from mapping reads against the provided reference genome. To speed up the program we will load the MDS decomposition `mash_mds.pkl` which was created by the COG analysis above:

```
pyseer --phenotypes resistances.pheno --vcf snps.vcf.gz --load-m mash_mds.pkl --lineage -
↪-print-samples > penicillin_SNPs.txt
```

This gives similar log messages:

```
Read 603 phenotypes
Detected binary phenotype
Loaded projection with dimension (603, 269)
Analysing 603 samples found in both phenotype and structure matrix
Writing lineage effects to lineage_effects.txt
198248 loaded variants
81370 filtered variants
116878 tested variants
116700 printed variants
```

We haven't specified the number of MDS dimensions to retain, so the default of 10 will be used (anything up to the 269 retained positive eigenvalues could be chosen). Turning on the test for lineage effects with `--lineage` uses the MDS components as the lineage, and writes the lineages most associated with the phenotype to `lineage_effects.txt`:

```
lineage wald_test       p-value
MDS3    10.3041807281   0.0
MDS10   6.61332035523   3.75794950713e-11
MDS5    6.03559150525   1.58381441295e-09
MDS4    2.35736678835   0.0184050574981
MDS6    1.33118701438   0.183127483126
```

(continues on next page)

(continued from previous page)

```
MDS2    1.02523510885   0.305252266
MDS9    0.850386297867  0.39511035157
MDS7    0.780676383001  0.434992854366
MDS1    0.478181602218  0.632520955891
MDS8    0.344928992152  0.730147754076
```

Variants associated with both the phenotype and MDS3, MDS10 or MDS5 may therefore be of interest as lineage effects.

The output now includes the lineage each variant is associated with, though not all variants can be assigned a lineage. `--print-samples` forces the inclusion of a comma separated list of samples the variant is present in `k-samples` and not present in `nk-samples` (not shown here for brevity):

```
variant af      filter-pvalue   lrt-pvalue      beta    beta-std-err    intercept      ␣
↪PC1      PC2     PC3     PC4     PC5     PC6     PC7     PC8 PC9      PC10    lineage␣
↪notes
26_23_G 4.31E-02        3.31E-01        4.42E-01        -4.19E-01       5.49E-01        -
↪9.22E-01        1.84E-01        -6.00E-01       -7.53E+00   8.84E-01 2.05E+01        -1.
↪79E+00        2.69E-01        1.16E-01        -7.52E-01       3.66E+00        MDS1
26_31_G_T       5.64E-02        3.94E-06        1.00E+00        6.78E-01        6.92E-01␣
↪        -8.90E-01       1.97E-01        -4.13E-01       -7.05E+00    8.63E-01        1.
↪91E+01         -1.33E+00       3.02E-01        9.13E-02        -4.99E-01       3.35E+00␣
↪        MDS10   bad-chisq
26_83_A_G       4.58E-01        9.88E-04        3.25E-01        4.06E-01        4.13E-01␣
↪        -1.21E+00       -1.43E-01       -7.84E-01       -7.35E+00    6.13E-01        1.
↪91E+01         -1.19E+00       1.73E-01        6.44E-01        -4.47E-01       3.63E+00␣
↪        MDS6
26_109_G_A      1.33E-02        1.46E-01        2.10E-14        4.15E+01        7.25E-01␣
↪        -9.97E-01       9.39E-02        3.33E-02        -9.52E+00    1.72E+00        3.
↪41E+01         1.38E+00        4.43E-01        -1.20E+00       6.82E-02        4.28E+00
26_184_G_A      3.32E-02        1.06E-02        8.49E-01        1.75E-01        9.11E-01␣
↪        -9.65E-01       1.37E-01        -5.96E-01       -7.42E+00    8.65E-01        1.
↪98E+01         -1.71E+00       3.00E-01        2.78E-01        -6.18E-01       3.63E+00
26_281_C_T      1.01E-01        1.20E-05        3.97E-01        -5.91E-01       6.91E-01␣
↪        -9.08E-01       1.12E-01        -7.04E-01       -7.24E+00    7.18E-01        2.
↪02E+01         -1.73E+00       4.32E-01        3.50E-01        -6.84E-01       3.69E+00␣
↪        MDS4
26_293_G_A      1.49E-02        3.50E-01        5.31E-01        7.06E-01        1.07E+00␣
↪        -9.73E-01       1.29E-01        -6.11E-01       -7.49E+00    9.16E-01        2.
↪03E+01         -1.54E+00       3.02E-01        2.55E-01        -5.93E-01       3.66E+00␣
↪        MDS6
26_483_G_A      2.37E-01        7.85E-02        1.82E-02        9.16E-01        3.90E-01␣
↪        -1.32E+00       -2.83E-01       -1.30E+00       -7.28E+00    6.77E-01        1.
↪78E+01         -1.79E+00       2.59E-01        1.10E+00        3.15E-02        3.44E+00␣
↪        MDS9
26_539_G_A      1.33E-02        1.46E-01        2.10E-14        4.15E+01        7.25E-01␣
↪        -9.97E-01       9.39E-02        3.33E-02        -9.52E+00    1.72E+00        3.
↪41E+01         1.38E+00        4.43E-01        -1.20E+00       6.82E-02        4.28E+00
```

This contains co-ordinates and p-values, which can be converted to a `.plot` file using the following `awk` one-liner:

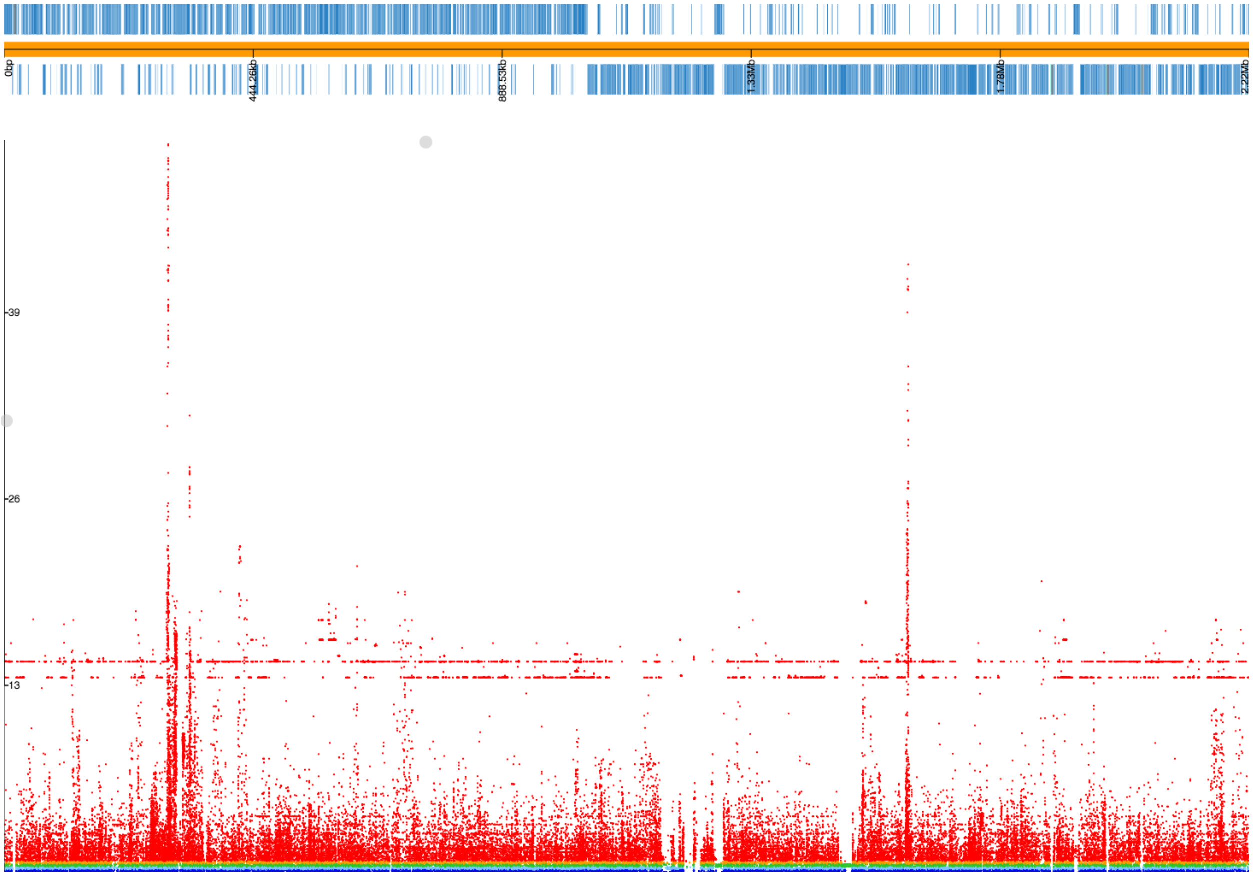
```
cat <(echo "#CHRSNPBPminLOG10(P)log10(p)r^2") \\
<(paste <(sed '1d' penicillin_SNPs.txt | cut -d "_" -f 2) \\
```

(continues on next page)

(continued from previous page)

```
<(sed '1d' penicillin_SNPs.txt | cut -f 4) | \\
awk '{p = -log($2)/log(10); print "26",".",$1,p,p,"0"}' ) | \\
tr ' ' '\t' > penicillin_snps.plot
```

If we drag and drop `23FSpn.gff` and `penicillin_snps.plot` files into phandango you should see a Manhattan plot similar to this:



The three highest peaks are in the *pbp2x*, *pbp1a* and *pbp2b* genes, which are the correct loci. There are also flat lines, suggesting these may be lineage effects from population structure that has not been fully controlled for. In actual fact, if we inspect the SNPs along these two lines (`p = 2.10E-14` and `p = 1.58E-15`) we see that all of them are annotated with the note `bad-chisq` and are at the lower end of the included minor allele frequency threshold (1.3% and 1.2% respectively). These are therefore variants which were underpowered, and the associations are spurious. They should be filtered out, and we should probably have used a MAF cutoff of at least 2% given the total number of samples we have. As a rule of thumb, a MAF cutoff corresponding to a MAC of at least 10 isn't a bad start. Let's run it again:

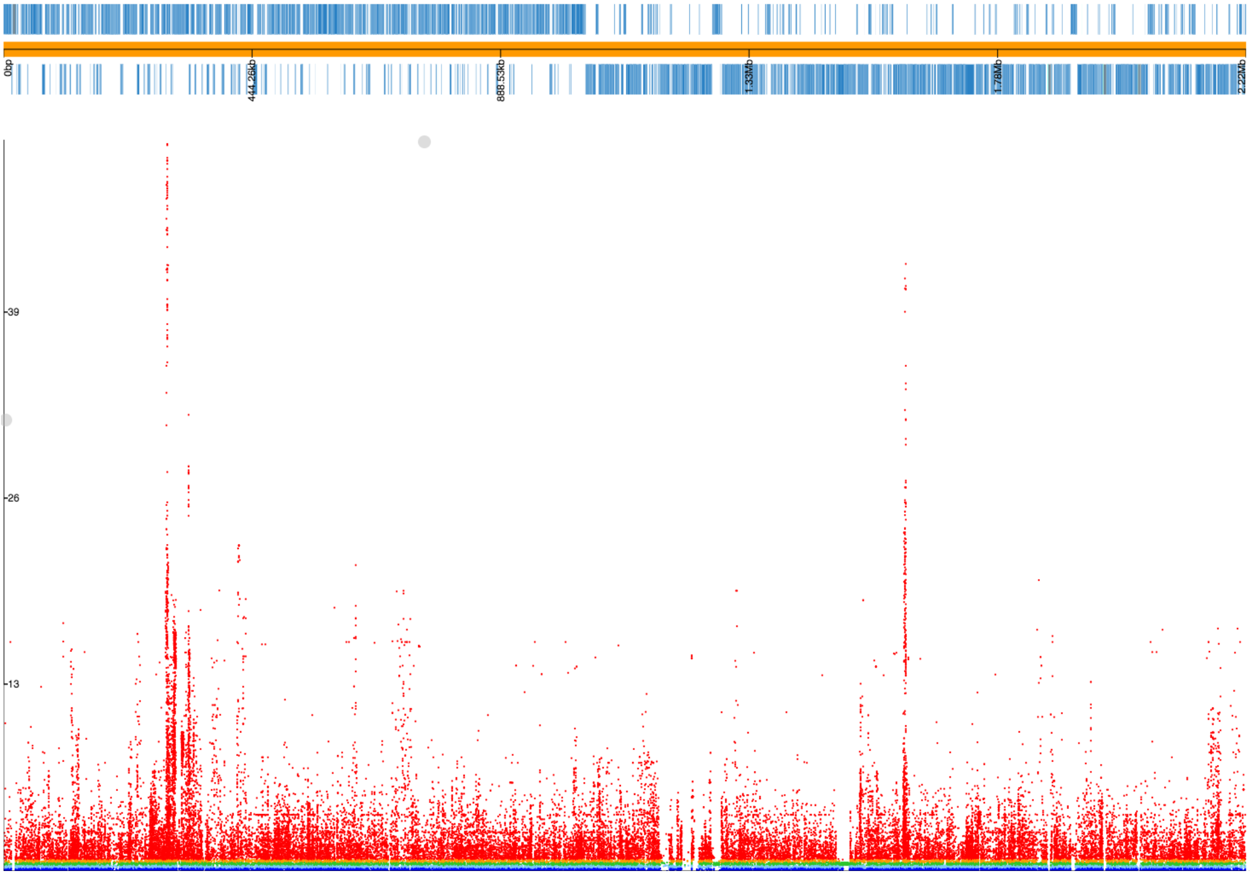
```
pyseer --phenotypes resistances.pheno --vcf snps.vcf.gz --load-m output/mash_mds.pkl --
↪min-af 0.02 --max-af 0.98 > penicillin_SNPs.txt

Read 603 phenotypes
Detected binary phenotype
Loaded projection with dimension (603, 269)
Analysing 603 samples found in both phenotype and structure matrix
198248 loaded variants
106949 filtered variants
91299 tested variants
```

(continues on next page)

(continued from previous page)

```
91225 printed variants
```

A lot more low frequency variants have been filtered out this time, and if we make a plot file our Manhattan plot looks much cleaner:



1.5.2 K-mer association with mixed effects model

We will now use k-mers as a variant to test both short variation as well as gene presence/absence. This can be done using the steps above replacing the `--vcf` argument with `--kmers`, which would replicate the results from the original `seer` tutorial. For demonstration purposes we will instead use the other association model available in `pyseer`, the linear mixed model.

First, count the k-mers from the assemblies:

```
mkdir -p assemblies
cd assemblies
tar xvf ../assemblies.tar.bz2
fsm-lite -l ../fsm_file_list.txt -s 6 -S 610 -v -t fsm_kmers | gzip -c - > ../fsm_kmers.
↪txt.gz
cd ..
```

This will require you to have fsm-lite installed If you do not have the time/resources to do this, you can follow the rest of these steps using the SNPs as above.

Note: Everything here also applies to unitigs, which can be called with unitig-counter. These are generally recommended due to their lower redundancy (and are also therefore faster) and potentially easier interpretation.

To correct for population structure we must supply `pyseer` with the kinship matrix K using the `--similarities` argument (or `--load-lmm` if using a previous analysis where `--save-lmm` was used).

We will use the distances from the core genome phylogeny, which has been midpointed rooted:

```
python scripts/phylogeny_distance.py --lmm core_genome_aln.tree > phylogeny_K.tsv
```

Note: Alternatively, we could extract a kinship matrix from the mapped SNPs by calculating $K = GG^T$ `similarity_pyseer --vcf snps.vcf.gz samples.txt > gg.snps.txt`

We can now run `pyseer` with `--lmm`. Due to the large number of k-mers we are going to test, we will increase the number of CPUs used to 8:

```
pyseer --lmm --phenotypes resistances.pheno --kmers fsm_kmers.txt.gz --similarity␣
↪phylogeny_K.tsv --output-patterns kmer_patterns.txt --cpu 8 > penicillin_kmers.txt
```

The heritability h^2 estimated from the kinship matrix K is printed to STDERR, and after about 5 hours the results have finished being written:

```
Read 603 phenotypes
Detected binary phenotype
Setting up LMM
Similarity matrix has dimension (616, 616)
Analysing 603 samples found in both phenotype and similarity matrix
h^2 = 0.90
15167239 loaded variants
1042215 filtered variants
14125024 tested variants
14124993 printed variants
```

Note: The heritability estimate shouldn't be interpreted as a quantitative measure for this binary phenotype, but a high heritability is consistent with the mechanism of penicillin resistance in this species (the sequence can give up to 99% prediction accuracy of penicillin resistance).

The results look similar, though also include the heritability of each variant tested:

```
variant af      filter-pvalue   lrt-pvalue      beta    beta-std-err    variant_h2      ␣
↪notes
TTTTTTTTTTTT    8.11E-01        1.51E-06        1.05E-01        6.13E-02        3.78E-02␣
↪        6.60E-02
TTTTTTTTTTTTT   7.08E-01        6.20E-06        4.03E-01        -3.34E-02       3.98E-02␣
↪        3.41E-02
TTTTTTTTTTTTTT  5.97E-01        6.39E-05        1.81E-01        -4.05E-02       3.03E-02␣
↪        5.45E-02
TTTTTTTTTTTTTTT 3.55E-01        5.92E-04        7.90E-01        -6.84E-03       2.57E-02␣
↪        1.09E-02
TTTTTTTTTTTTTTTT        1.48E-01        2.11E-03        7.38E-01        1.13E-02        ␣
```

(continues on next page)

(continued from previous page)

```
↪3.37E-02        1.37E-02
TTTTTTTTTTTTTTTTT       6.47E-02        3.94E-01        4.89E-01        3.11E-02        ␣
↪4.49E-02        2.83E-02
TTTTTTTTTTTTTTTTTT      3.48E-02        2.73E-02        2.59E-01        -6.73E-02       ␣
↪5.96E-02        4.60E-02
TTTTTTTTTTTTTTTTTTT     2.32E-02        2.18E-01        6.96E-01        -2.81E-02       ␣
↪7.19E-02        1.59E-02
TTTTTTTTTTTTTTTTTTTT    1.66E-02        2.58E-01        9.46E-01        -5.63E-03       ␣
↪8.37E-02        2.74E-03
```

The downstream processing of the k-mer results in `penicillin_kmers.txt` will be shown in the next section. Before that, we can determine a significance threshold using the number of unique k-mer patterns:

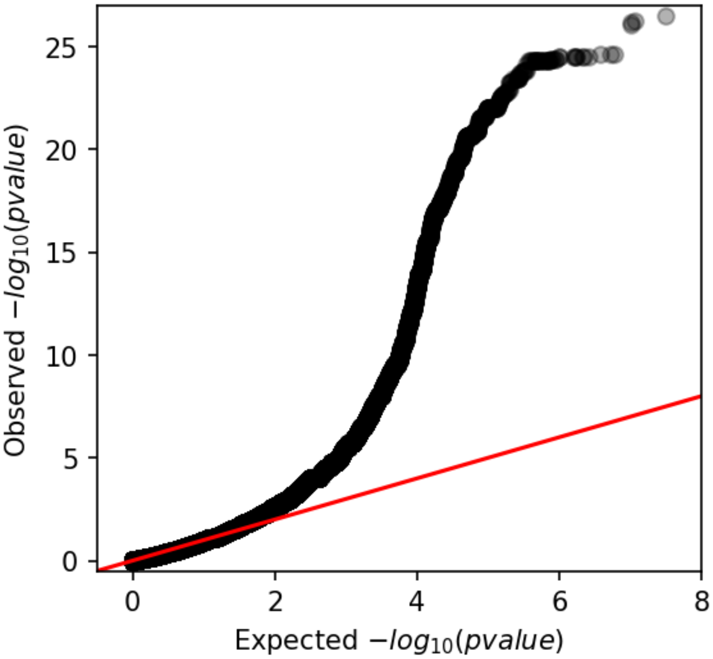
```
python scripts/count_patterns.py kmer_patterns.txt
Patterns:       2627332
Threshold:      1.90E-08
```

This is over five times lower than the total number of k-mers tested, so stops us from being hyper-conservative with the multiple testing correction.

We can also create a Q-Q plot to check that p-values are not inflated. We can do that by using the `qq_plot.py` script:

```
python scripts/qq_plot.py penicillin_kmers.txt
```

which produces the following Q-Q plot:



When interpreting this plot, check that it is well controlled at low p-values and doesn't show any large 'shelves' symptomatic of poorly controlled confounding population structure. Although this plot is far above the null (as indeed, there are many k-mers associated with penicillin resistance), the p-values up to 0.01 are as expected which is what we're after.

1.5.3 Interpreting significant k-mers

For the final step we will work with only those k-mers which exceeded the significance threshold in the mixed model analysis. We will filter these from the output using a simple awk command:

```
cat <(head -1 penicillin_kmers.txt) <(awk '$4<1.90E-08 {print $0}' penicillin_kmers.txt) > significant_kmers.txt
```

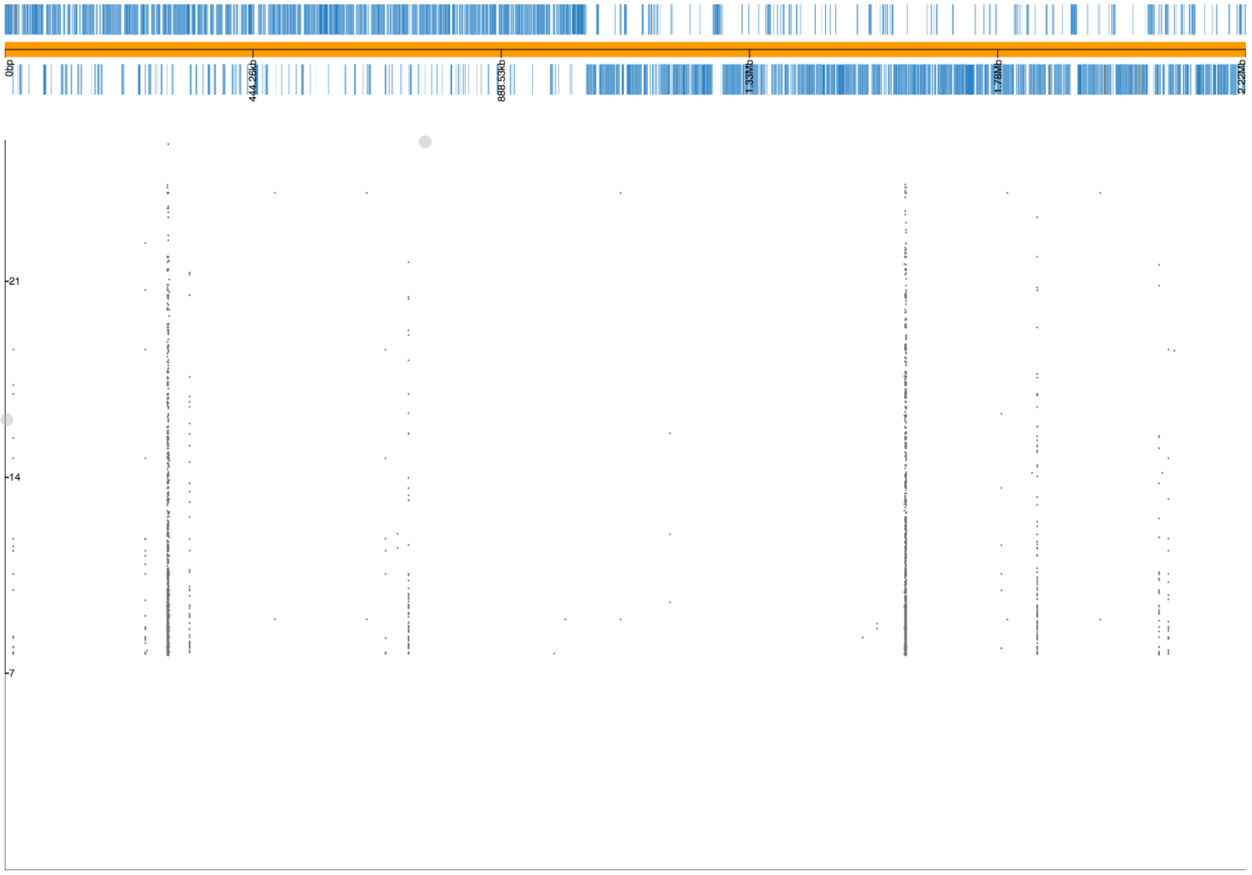
There are 5327 significant k-mers.

Mapping to a single reference

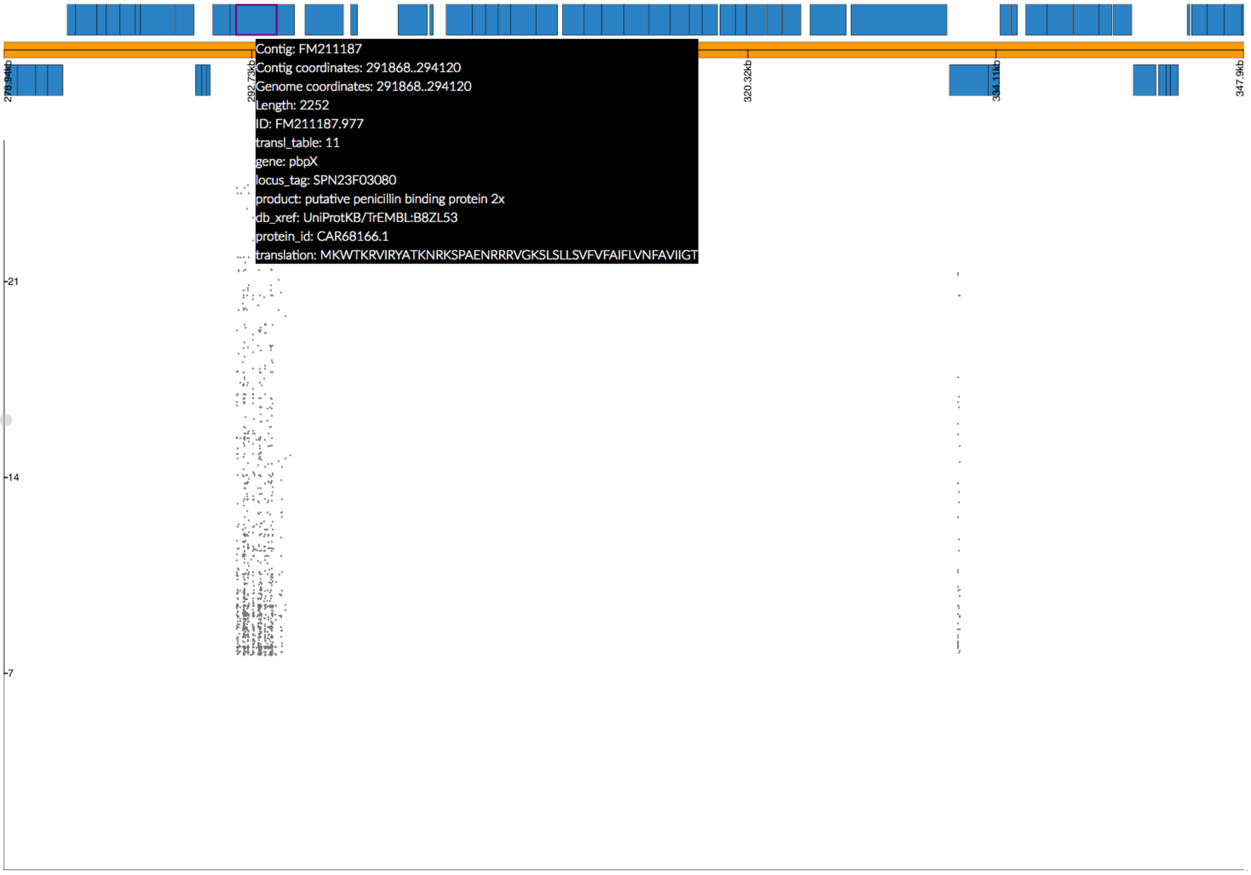
Let's use bwa mem to map these to the reference provided:

```
phandango_mapper significant_kmers.txt Spn23F.fa Spn23F_kmers.plot

Read 5327 k-mers
Mapped 2425 k-mers
```

Not all the k-mers have been mapped, which is usually the case. Note there are 2459 mapping lines in the output, as 34 secondary mappings we included. It is a good idea to map to range of references to help with an interpretion for all of the significant k-mers. The k-mer annotation step, described next, also helps cover all k-mers. Let's look at the plot file in phandango:



In this view we no longer see all of the Manhattan plot as we have filtered out the low p-value k-mers. There is generally less noise due to LD/population structure when compared to our previous result above. There are peaks in the three *pbp* genes again, with the strongest results in *pbp2x* and *pbp2b* as before. Zooming in:



Contig: FM211187
Contig coordinates: 291868..294120
Genome coordinates: 291868..294120
Length: 2252
ID: FM211187.977
transl_table: 11
gene: pbpX
locus_tag: SPN23F03080
product: putative penicillin binding protein 2x
db_xref: UniProtKB/TrEMBL:B8ZL53
protein_id: CAR68166.1
translation: MKWTKRVIRYATKNRKSPAENRRRVGKSLSLLSVFVFAIFLVNFAVIIGT

The whole *pbp2x* gene is covered by significant k-mers, whereas only a small part of *pbp1a* is hit. This could be due to the fact that only some sites in *pbp1a* can be variable, only some of the variable sites affect penicllin resistance, or due to the ability to map k-mers to this region.

Annotating k-mers

We can annotate these k-mers with the genes they are found in, or are near. To try and map every k-mer we can include a number of different reference annotations, as well as all the draft annotations of the sequences the k-mers were counted from. For the purposes of this tutorial we will demonstrate with a single type of each annotation, but this could be expanded by adding all the annotated assemblies to the input.

We'll start by creating a `references.txt` file listing the annotations we wish to use (see *Annotating k-mers* for more information on how to construct this file):

```
Spn23F.fa    Spn23F.gff     ref
6952_7#3.fa  6952_7#3.gff   draft
```

Now run the script. This will iterate down the list of annotations, annotating the k-mers which haven't already been mapped to a previous annotation (requires `bedtools`, `bedops` and the `pybedtools` package):

```
annotate_hits_pyseer significant_kmers.txt references.txt annotated_kmers.txt

Reference 1
```

(continues on next page)

(continued from previous page)

```
5327 kmers remain
Draft reference 2
2902 kmers remain
```

Note: If this runs slowly you can split the significant_kmers.txt file into pieces to parallelise the process.

Note: By default annotate_hits_pyseer will only consider CDS features in the provided GFF files. If you want to consider other feature types you can use the --feature-type option (*e.g.* --feature-type rRNA --feature-type tRNA).

Annotations marked ref can partially match between k-mer and reference sequence, whereas those marked draft require an exact match. In this case the single draft didn't add any matches. The genes a k-mer is in, as well as the nearest upstream and downstream are added to the output:

```
TTTTTTTCTACAATAAAATAGGCTCCATAATATCTATAGTGGATTTACCCACTACAAATATTATAGAACCCGTTTTATTATGGAAAGACTTATTGGACTT␣
↪   6.47E-02        2.08E-12        2.10E-09        7.97E-01        1.31E-01        2.
↪41E-01        FM211187:252213-252312;FM211187.832;;FM211187.834
TTTTTTTATAGATTTCAGGATCAGCCAAATAGTAATCCG 8.42E-01        1.03E-36        2.99E-10        -
↪4.38E-01       6.83E-02        2.53E-01        FM211187:723388-723417;FM211187.2367;;
↪FM211187.2371
TTTTTTTATAGATTTCAGGATCAGCCAAATAGTAATCCGCCAGCTGGCGTT     8.39E-01        3.38E-35        ␣
↪4.04E-09       -3.95E-01       6.62E-02        2.37E-01        FM211187:1614084-
↪1614122;penA;penA;penA
```

The output format is contig:position;upstream;in;downstream. The first line shows the k-mer was mapped to FM211187:252213-252312, the nearest gene downstream having ID FM211187.832 and upstream having ID FM211187.834. The third line shows that k-mer overlaps *penA* – note when a gene= field is found this is used in preference to the ID= field.

Finally, we can summarise these annotations to create a plot of significant genes. We will only use genes k-mers are actually in, but if we wanted to we could also include up/downstream genes by using the --nearby option:

```
python scripts/summarise_annotations.py annotated_kmers.txt > gene_hits.txt
```

We'll use ggplot2 in R to plot these results:

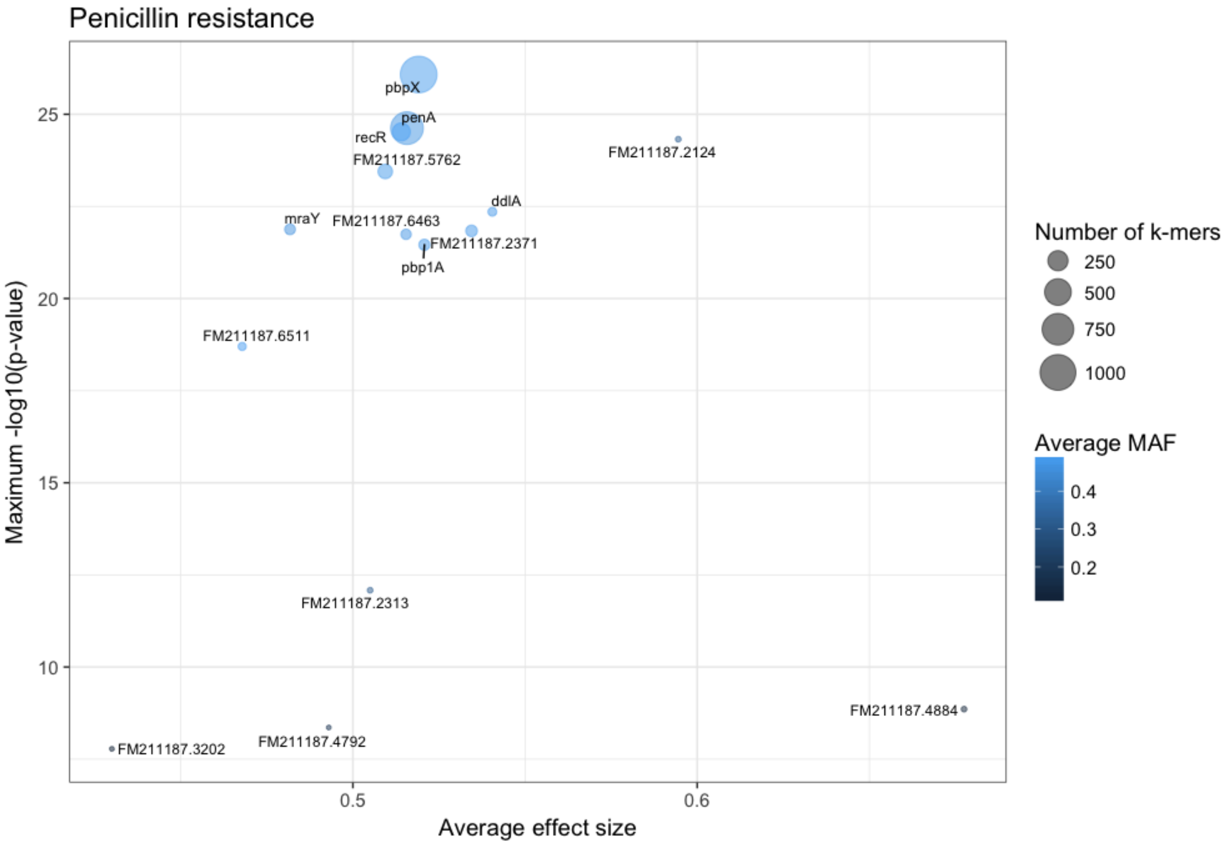
```
require(ggplot2)
require(ggrepel)
library(ggrepel)

gene_hits = read.table("gene_hits.txt", stringsAsFactors=FALSE, header=TRUE)

ggplot(gene_hits, aes(x=avg_beta, y=maxp, colour=avg_maf, size=hits, label=gene)) +
   geom_point(alpha=0.5) +
   geom_text_repel(aes(size=60), show.legend = FALSE, colour='black') +
   scale_size("Number of k-mers", range=c(1,10)) +
   scale_colour_gradient('Average MAF') +
   theme_bw(base_size=14) +
   ggtitle("Penicillin resistance") +
   xlab("Average effect size") +
```

(continues on next page)

(continued from previous page)

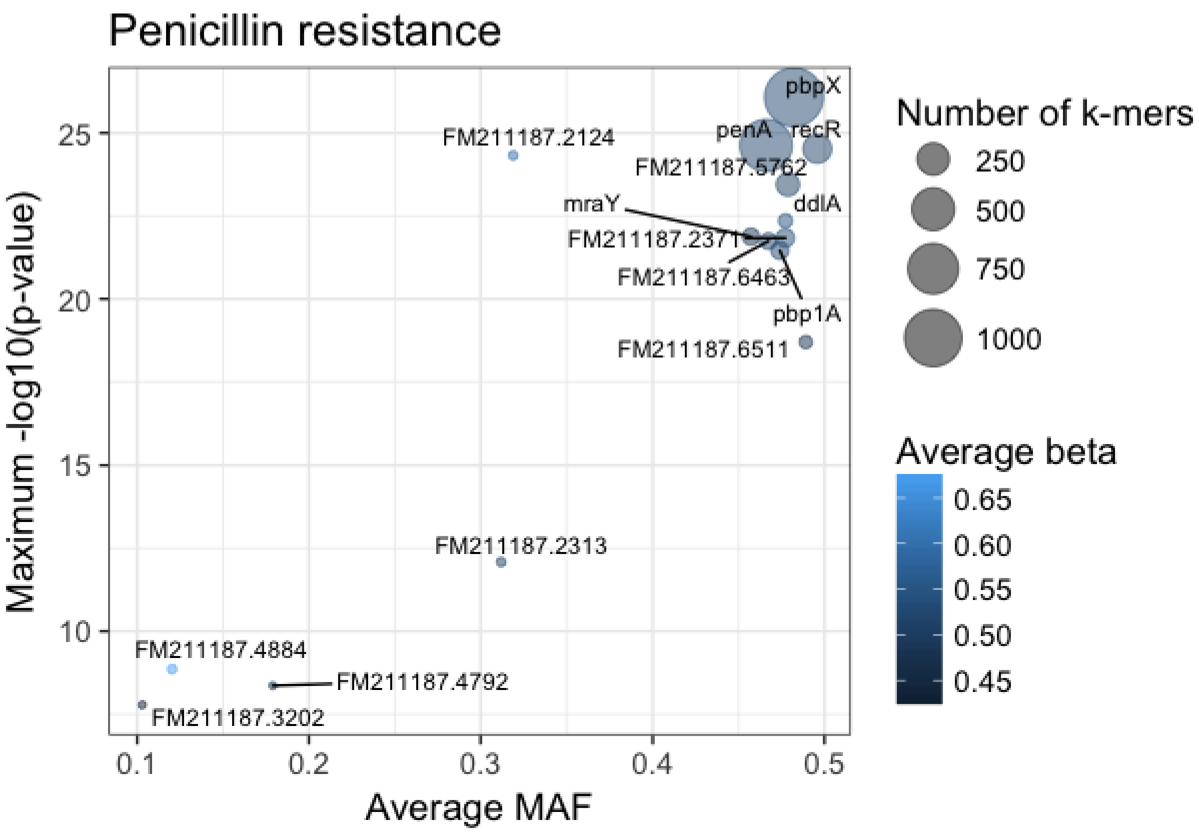
```
  ylab("Maximum -log10(p-value)")
```

You can customise this however you wish (for example adding the customary italics on gene names); these commands will produce a plot like this:



The main hits have high p-values and are common, and in this case are covered by many k-mers. In this case *penA* (*pbp2b*) and *penX* (*pbp2x*) are the main hits. Other top genes *recR* and *ddl* are adjacent to the *pbp* genes and are in LD with them, creating an artifical association.

The results with large effect sizes (see *Effect sizes*) and relatively low p-values also have low MAF, and are probably false positives. This can be seen better by changing the axes:



1.6 Prediction tutorial

This page describes how to fit whole genome models with `--wg`, and how they can be used to predict the phenotype for new samples. This tutorial starts from the same dataset as the *GWAS tutorial*, which is described at the top of that page.

Note: Presently only the elastic net is implemented, which is the method used in this tutorial. Future methods will include random forests and best linear unbiased predictors (BLUPs).

1.6.1 Fitting a whole-genome model

The first step to performing prediction is to a train a model on genetic data with a known phenotype. The trained models in pyseer can also be used for association purposes, as the individual variants associated with the phenotype are reported.

Here we will try and find SNPs which can predict penicillin resistance in *S. pneumoniae*. It would also be possible to use unitigs by changing `--vcf` to `--kmers`. We will use the same data as from the GWAS *GWAS tutorial* – instructions on how to download this data can be found at the top of that page.

Variants are loaded, the model is fitted and saved. This can all be done in a single step:

```
pyseer --vcf snps.vcf.gz --phenotypes resistances.pheno --wg enet \
--save-vars output/ma_snps --save-model penicillin.lasso --cpu 4 --alpha 1 > selected.txt

Read 603 phenotypes
Detected binary phenotype
Reading all variants
198248variants [04:46, 691.24variants/s]
Saved enet variants as output/ma_snps.pkl
Applying correlation filtering
100%|| 89703/89703 [00:51<00:00, 1742.25variants/s]
Fitting elastic net to top 67277 variants
[status]    Parallel glmnet cv with 4 cores
Best penalty (lambda) from cross-validation: 2.09E-02
Best model deviance from cross-validation: 0.405 ± 4.57E-02
Best R^2 from cross-validation: 0.822
Finding and printing selected variants
Saved enet model as penicillin.lasso.pkl
198248 loaded variants
130971 filtered variants
67277 tested variants
35 printed variants
```

> **Warning:** You may see warnings about variants with no observations. In this case the VCF has many missing calls causing this, which can be ignored. In other settings this often points to a mismatch between sample labels in the variant and phenotype files.

`selected.txt` now contains the selected variants in a GWAS-like format:

```
variant     af      filter-pvalue   lrt-pvalue      beta    notes
26_31771_C_T        3.48E-02        8.06E-02                1.10E-01
26_292628_G_A       3.78E-01        6.12E-94                9.92E-02
26_292653_T_C       6.63E-02        2.95E-16                9.56E-01        bad-chisq
```

To calculate an adjusted p-value you can add `--distances` as one would do for GWAS with the SEER fixed effects model, or create a new variant file with just the selected variants, then run pyseer again.

Differences of this approach from the univariate GWAS approach covered in *GWAS tutorial*:

- `--wg enet` fits an elastic net to all variants with `--n-folds` cross-validation (default 10-fold).
- `--save-vars` saves the variants loaded by `--vcf` in an efficient sparse matrix format, which can be quickly loaded for new model fitting.

- `--save-model` saves the fitted model so it can be used for prediction.
- `--cpu` uses four cores efficiently during cross-valdation.

`--alpha` controls the mixing between ridge regression and lasso regression. Above we have used a value of 1, which is lasso regression, selecting just a few variants. We can use a value closer to ridge regression if desired, which will select more variants with smaller effect sizes:

```
pyseer --vcf snps.vcf.gz --phenotypes resistances.pheno --wg enet \
--load-vars output/ma_snps --save-model penicillin.001 --alpha 0.01 > selected.txt

Read 603 phenotypes
Detected binary phenotype
Reading all variants
Analysing 603 samples found in both phenotype and loaded npy
Applying correlation filtering
100%|| 89703/89703 [01:03<00:00, 1421.87variants/s]
Fitting elastic net to top 67275 variants
Best penalty (lambda) from cross-validation: 8.26E-01
Best model deviance from cross-validation: 0.402 ± 4.45E-02
Best R^2 from cross-validation: 0.815
Finding and printing selected variants
Saved enet model as penicillin.001.pkl
198248 loaded variants
130973 filtered variants
67275 tested variants
3523 printed variants
```

We can load the variants saved previously which saves a lot of time. The variant file is needed to print the selected variants at the end – this is checked to ensure it is the same as the one originally provided.

Loading the variants can also be used when just a subset of `--phenotypes` is provided, which is useful for training-test validation.

Accounting for population structure

As the model includes all genetic variants at once, covariance between them from population structure can implicitly be included already. However, it is possible to include an explicit correction for population structure which may improve prediction accuracy in new populations.

This correction is based on providing discrete definitions of lineages/strains. Prepare a file `lineages.txt` with the following format:

```
7001_3#17   0
6999_7#9    0
7622_5#50   0
6999_1#2    0
7622_4#1    0
...
7622_2#40   59
7622_3#86   60
7622_5#61   61
```

Important: Rare lineages must be represented correctly, i.e. in their own cluster rather than being grouped in a 'bin'. One method we recommend to do this is PopPUNK. Connecting samples together which are below a certain distance threshold will also work.

If you need to convert from PopPUNK output to this format with the tutorial, you can use the following code:

```
import csv, re
reader = csv.DictReader(open("poppunk/poppunk_clusters.csv"))
writer = csv.DictWriter(open("lineages.txt", "w"), delimiter=' ', fieldnames=reader.
↪fieldnames)
for row in reader:
    row['Taxon'] = re.match(r'.*/(.*)\.contigs_velvet\.fa', row['Taxon']).group(1)
    writer.writerow(row)
```

Now add this to the analysis:

```
pyseer --vcf snps.vcf.gz --phenotypes resistances.pheno --wg enet \
--load-vars output/ma_snps --lineage-clusters lineages.txt --sequence-reweighting

Read 603 phenotypes
Detected binary phenotype
Reading all variants
Analysing 603 samples found in both phenotype and loaded npy
Applying correlation filtering
100%|| 89703/89703 [00:59<00:00, 1513.70variants/s]
Fitting elastic net to top 67275 variants
Fitting elastic net to top 67275 variants
Best penalty (lambda) from cross-validation: 1.17E+00
Best model deviance from cross-validation: 0.572 ± 8.76E-02
Best R^2 from cross-validation: 0.815
Predictions within each lineage
Lineage     Size    R2      TP      TN      FP      FN
0   96      0.820   35      57      4       0
1   55      0.182   2       48      0       5
...
8   18      -0.200  0       15      0       3
9   18      1.000   0       18      0       0
Finding and printing selected variants
198248 loaded variants
130973 filtered variants
67275 tested variants
4357 printed variants
```

Adding `--lineage-clusters` has two effects. Cross-validation will be performed by leaving one strain out. This will usually take longer as there are more strains than folds, but may help reduce the number of lineage effects included. Also, training predition accuracy for each lineage will be reported, making it easier to see whether there are some parts of the data where the model is performing better. For binary phenotypes R^2 can be difficult to interpret, so true/false positives/negatives are also reported.

Adding `--sequence-reweighting` has one further effect. Within each lineage, the weight w_i given to each sample in the loss function

$$\min_{b_0,b} \frac{1}{N} \sum_{i=1}^{N} w_i l(y_i, b_0 + b^T x_i)^2 + \lambda \left[(1-\alpha)||b||_2^2/2 + \alpha||b||_1 \right]$$

is set by

$$\frac{1}{u_i} = \sum_{j=1}^{N} [j \in C(i)]$$

$$w_i = u_i \cdot \frac{N}{\sum_{j=1}^{N} u_j}$$

where $C(x)$ is the lineage cluster of x.

This sets the weights as being inversely proportional to the size of the cluster, and rescales all weights to sum to N. Without this option $w_i = 1 \; \forall \; i$.

1.6.2 Using the model to predict phenotype in new samples

The elastic net models can be used to predict phenotypes in new samples. We will first split the samples into training and test sets:

```
head -500 resistances.pheno > train.pheno
cat <(head -1 resistances.pheno) <(tail -104 resistances.pheno) > test.pheno
cut -f 1 test.pheno | sed '1d' > test.samples
```

Warning: This is a random split of the samples, unlikely to be equivalent to different sample collections made up of different proportions of strains. Accuracy is likely overestimated, but within strain accuracies can be useful.

We will use lasso regression as fewer variants are selected, so if they were uncalled in the test set this should be less of a problem (but is still an important concern). Fit a model to the training set:

```
pyseer --vcf snps.vcf.gz --phenotypes train.pheno --wg enet \
--load-vars output/ma_snps --alpha 1 --save-model test_lasso --cpu 4 \
--lineage-clusters lineages.txt --sequence-reweighting

Read 499 phenotypes
Detected binary phenotype
Reading all variants
Analysing 499 samples found in both phenotype and loaded npy
Applying correlation filtering
100%|| 89703/89703 [00:56<00:00, 1597.01variants/s]
Fitting elastic net to top 67277 variants
[status]    Parallel glmnet cv with 4 cores
Best penalty (lambda) from cross-validation: 3.38E-02
Best model deviance from cross-validation: 0.605 ± 1.01E-01
Best R^2 from cross-validation: 0.788
Predictions within each lineage
Lineage     Size    R2      TP      TN      FP      FN
0   74      0.753   24      46      4       0
1   41      0.219   2       35      0       4
10  12      1.000   0       12      0       0
11  9       1.000   8       1       0       0
12  8       1.000   8       0       0       0
13  11      1.000   11      0       0       0
14  9       1.000   3       6       0       0
```

(continues on next page)

(continued from previous page)

```
15  9       1.000   0       9       0       0
16  10      1.000   0       10      0       0
17  7       -0.167  0       6       0       1
18  6       1.000   0       6       0       0
19  5       -0.250  0       4       0       1
2   35      1.000   0       35      0       0
20  3       1.000   3       0       0       0
21  6       -0.200  0       5       0       1
22  7       1.000   0       7       0       0
23  6       1.000   0       6       0       0
24  7       -0.167  0       6       0       1
25  7       1.000   0       7       0       0
26  6       -0.200  0       5       0       1
27  5       1.000   0       5       0       0
28  5       1.000   2       3       0       0
29  5       1.000   5       0       0       0
3   36      -0.059  34      0       2       0
30  3       1.000   0       3       0       0
31  4       -0.333  0       3       0       1
32  4       1.000   0       4       0       0
33  3       -0.500  0       2       0       1
34  3       1.000   3       0       0       0
35  3       1.000   0       3       0       0
36  3       1.000   0       3       0       0
37  3       1.000   3       0       0       0
38  1       1.000   0       1       0       0
39  1       1.000   0       1       0       0
4   24      -0.043  23      0       1       0
40  2       1.000   2       0       0       0
41  2       1.000   0       2       0       0
42  1       1.000   0       1       0       0
43  2       1.000   0       2       0       0
44  1       1.000   0       1       0       0
45  1       1.000   0       1       0       0
46  2       1.000   0       2       0       0
47  1       1.000   1       0       0       0
48  1       1.000   0       1       0       0
49  1       1.000   0       1       0       0
5   24      1.000   24      0       0       0
50  1       1.000   0       1       0       0
51  1       1.000   1       0       0       0
52  1       1.000   0       1       0       0
53  1       1.000   1       0       0       0
54  1       1.000   1       0       0       0
55  1       1.000   1       0       0       0
56  1       1.000   1       0       0       0
57  1       1.000   0       1       0       0
58  1       1.000   0       1       0       0
59  1       1.000   0       1       0       0
6   18      -0.059  0       17      0       1
7   18      -0.200  12      3       0       3
8   18      -0.200  0       15      0       3
```

(continues on next page)

(continued from previous page)

```
9   16      1.000   0       16      0       0
Finding and printing selected variants
Saved enet model as test_lasso.pkl
198248 loaded variants
130971 filtered variants
67277 tested variants
32 printed variants
```

The prediction accuracy is pretty similar across lineages, which is good. As the test set is a similar makeup of lineages hopefully prediction accuracy will be similar.

`enet_predict` is used to make the predictions:

```
enet_predict --vcf snps.vcf.gz --lineage-clusters lineages.txt --true-values test.pheno \
test_lasso.pkl test.samples > test_predictions.txt

Reading variants from input
198248variants [00:11, 17657.99variants/s]
Overall prediction accuracy
R2: 0.8668373879641486
tn: 69
fp: 2
fn: 1
tp: 32
Predictions within each lineage
Lineage     Size    R2      TP      TN      FP      FN
0   22      1.000   11      11      0       0
1   14      -0.077  0       13      0       1
10  3       1.000   0       3       0       0
11  5       1.000   2       3       0       0
12  5       -0.250  4       0       1       0
13  1       1.000   1       0       0       0
14  2       1.000   1       1       0       0
15  2       1.000   0       2       0       0
17  2       1.000   0       2       0       0
18  2       1.000   0       2       0       0
19  4       1.000   0       4       0       0
2   11      1.000   0       11      0       0
20  4       -0.333  3       0       1       0
21  1       1.000   0       1       0       0
23  1       1.000   0       1       0       0
26  1       1.000   0       1       0       0
27  1       1.000   0       1       0       0
3   8       1.000   8       0       0       0
30  1       1.000   0       1       0       0
33  1       1.000   0       1       0       0
39  1       1.000   0       1       0       0
4   1       1.000   1       0       0       0
42  1       1.000   0       1       0       0
44  1       1.000   0       1       0       0
45  1       1.000   0       1       0       0
5   1       1.000   1       0       0       0
6   2       1.000   0       2       0       0
```

(continues on next page)

(continued from previous page)

```
60  1       1.000   0       1       0       0
61  1       1.000   0       1       0       0
7   1       1.000   0       1       0       0
9   2       1.000   0       2       0       0
```

The required options are a variant file, in this case the same `--vcf` contains calls for the test samples, but this could be a new file, as long as the variant labels match (non-trivial!). `test_lasso.pkl` is the saved model and `test.samples` are the names of samples appearing in the variants file to produce predictions for.

Here, providing `--true-values` is needed to give the prediction accuracies. Providing `--lineage-clusters` in addition gives the per lineage prediction accuracy. For the reasons noted above, the test accuracy is pretty similar to the training set.

The predictions are in `test_predictions.txt`:

```
Sample  Prediction      Link    Probability
7622_3#79       1.0     1.1723387708055686      0.7635674993396665
7622_3#80       1.0     2.828167499490956       0.9441790988402875
7622_3#81       1.0     2.2308130622857987      0.9029826106893201
7622_3#82       0.0     -0.7572524088945985     0.3192430949937001
```

For a binary phenotype:

- a 0/1 prediction at `--threshold` on the probability.
- Link is the value of the linear sum of the model betas, before entering the logit link function.
- Probability is a continuous prediction (after taking logit).

Generating consistent unitig calls

It has been mentioned many times above that it is necessary that variant calls match between the inputs of the training and test data. This was ensured above as all variants were called together and merged into a single file. Generally this may not be possible, especially if testing prediction accuracy in a new cohort. If a variant in the model is missing its mean slope value will be used for all samples, which may significantly reduce accuracy.

One way around this issue is to use unitigs. However, sequences which are unitigs in the DBG of one population may not be unitigs in the DBG of a different sample set, even if they are present. So simply running `unitig-counter` on both training and test datasets will result in many missing calls.

You should instead use unitig-caller to make variant calls in the test population using the same unitigs definitions as in the training population. Full usage and details are given in the `README.md`, but briefly:

```
gzip -d -c unitigs.txt.gz | cut -f 1 > queries.txt
unitig-caller --mode simple --strains strain_list.txt --unitigs queries.txt --output␣
↪calls.txt
```

Will write a file of sequence elements for the samples in `strain_list.txt` to `calls.txt`, which is guaranteed to overlap with the original training set calls, and can therefore be used with `enet_predict`.

1.7 Multiprocessing

`pyseer` supports the use of multiple CPUs through the `--cpu` option. This sends batches of processed variants to a core, which will fit the chosen model on all variants in the batch.

The constant `--block-size` controls the number of variants sent to each core. The higher this is set the more efficient the use of CPUs will be (up to a limit, set by the time spent reading the variant input) at the expense of a roughly linear increase in memory usage. The default is 1000, using which on 8 cores required around 1.5Gb of memory for a 1.4x speedup with the mixed model. Increasing this to 30000 while using 4 cores gave a similar (1.5x) speedup, but needed 12Gb of memory.

Depending on your computing architecture, you may wish to split the input and run separate jobs. This will be more efficient, but is less convenient. This can be done using GNU `split`:

```
split -d -n l/8 fsm_kmers.txt fsm_out
```

This would split the input k-mers into 8 separate files.

1.7.1 Prediction

The `--wg enet` mode also supports CPUs, but can be very memory-hungry (memory use scales linearly with number of cores). For large datasets, if you are running out of memory, you may wish to try with just a single core.

1.8 Reference documentation

1.8.1 input.py

Functions to read data into pyseer and iterate over instances

`pyseer.input.`**`file_hash`**(*filename*)

Calculates the hash of an entire file on disk

Parameters
`filename` (*`str`*) – Location of file on disk

Returns

hash (str)
SHA256 checksum

`pyseer.input.`**`hash_pattern`**(*k*)

Calculates the hash of a presence/absence vector

Parameters
`k` (*`numpy.array`*) – Variant presence/absence binary vector (n, 1)

Returns

hash (byte)
Hashed pattern

`pyseer.input.`**`iter_variants`**(*p*, *m*, *cov*, *var_type*, *burden*, *burden_regions*, *infile*, *all_strains*, *sample_order*, *lineage_effects*, *lineage_clusters*, *min_af*, *max_af*, *max_missing*, *filter_pvalue*, *lrt_pvalue*, *null_fit*, *firth_null*, *uncompressed*, *continuous*)

Make an iterable to pass single variants to fixed effects regression

Parameters

- `p` (*pandas.DataFrame*) – Phenotype vector (n, 1)
- `m` (*numpy.array*) – Population structure matrix (n, k)
- `cov` (*pandas.DataFrame*) – Covariates matrix (n, m)
- `var_type` (*str*) – Variants type (one of: kmers, vcf or Rtab)
- `burden` (*bool*) – Whether to slice a vcf file by burden regions
- `burden_regions` (*collections.deque*) – Burden regions to slice the vcf with
- `infile` (*opened file*) – Handle to opened variant file
- `all_strains` (*set-like*) – All sample labels that should be present
- `sample_order` – Sampes order to interpret each Rtab line
- `lineage_effects` (*bool*) – Whether to fit lineage effects
- `clusters` (*lineage*) – Lineage clusters indexes
- `min_af` (*float*) – Minimum allele frequency (inclusive)
- `max_af` (*float*) – maximum allele frequency (inclusive)
- `max_missing` (*float*) – maximum missing frequency
- `filter_pvalue` (*float*) – Pre-filtering p-value threshold
- `lrt_pvalue` (*float*) – Filtering p-value threshold
- `null_fit` (*float or statsmodels.regression.linear_model.RegressionResultsWrapper*) – Null-fit likelihood (binary) or model (continuous)
- `firth_null` (*float*) – Firth regression likelihood
- `uncompressed` (*bool*) – Whether the kmers file is uncompressed
- `continuous` (*bool*) – Whether the phenotype is continuous or not

Returns

var_name (str)
Variant name

v (numpy.array)
Phenotypes vector (n, 1)

k (numpy.array)
Variant presence/absence vector (n, 1)

m (numpy.array)
Population structure matrix (n, k)

c (numpy.array)
Covariates matrix (n, m)

af (float)
Allele frequency

pattern (bytes)
Variant hash

lineage_effects (bool)
Whether to fit lineage effects

lineage clusters (list)
Lineage clusters indexes

filter_pvalue (float)
Pre-filtering p-value threshold

lrt_pvalue (float)
Filtering p-value threshold

null_fit (float or statsmodels.regression.linear_model.RegressionResultsWrapper)
Null-fit likelihood (binary) or model (continuous)

firth_null (float)
Firth regression likelihood

kstrains (iterable)
Sample labels with the variant

nkstrains (iterable)
Sample labels without the variant

continuous (bool)
Whether the phenotype is continuous or not

`pyseer.input.`**`iter_variants_lmm`**`(`*variant_iter*, *lmm*, *h2*, *lineage*, *lineage_clusters*, *covariates*, *continuous*, *filter_pvalue*, *lrt_pvalue*`)`

Make an iterable to pass single variants to fixed effects regression

`pyseer.input.`**`load_burden`**`(`*infile*, *burden_regions*`)`

Load burden regions for VCF analysis

Parameters

- **`infile`** (*`str`*) – Input file for burden regions
- **`burden_regions`** (*`list`*) – List to be filled in-place

`pyseer.input.`**`load_covariates`**`(`*infile*, *covariates*, *p*`)`

Load and encode a covariates matrix

Parameters

- **`infile`** (*`str`*) – Input file for the covariates matrix
- **`covariates`** (*`iterable or None`*) – List of string indicating which columns to use and their interpretation. Example: *2q* indicates that the second column from the file is a quantitative variable, *2* indicates that that same column is categorical. If None, the matrix is loaded but nothing is done with it.
- **`p`** (*`pandas.Series`*) – Phenotypes vector (n, 1)

Returns

cov (pandas.DataFrame)
Covariance matrix (n, m)

`pyseer.input.`**`load_lineage`**`(`*infile*, *p*`)`

Load custom lineage clusters definitions

Parameters

- **`infile`** (*`str`*) – Input file for lineage clusters
- **`p`** (*`pandas.Series`*) – Phenotypes vector (n, 1)

Returns

result (tuple of (numpy.array, list))
Lineage binary matrix and cluster labels

`pyseer.input.`**`load_phenotypes`**(*infile*, *column*)

Load phenotypes vector

Parameters

- **`infile`** (*`str`*) – Matrix input file
- **`column`** (*`str or None`*) – Phenotype column name or None to pick the last column

Returns

p (pandas.Series)
Phenotype vector (n, 1)

`pyseer.input.`**`load_structure`**(*infile*, *p*, *max_dimensions*, *mds_type='classic'*, *n_cpus=1*, *seed=None*)

Load population structure and apply multidimensional scaling

Parameters

- **`infile`** (*`str`*) – Population structure (distance matrix) input file
- **`p`** (*`pandas.Series`*) – Phenotype vector (n, 1)
- **`max_dimensions`** (*`int`*) – Maximum dimensions to consider when applying *metric* or *non-metric* MDS
- **`mds_type`** (*`str`*) – MDS algorithm to apply. One of *classic*, *metric* or *non-metric*. Any other input will trigger the *metric* MDS
- **`n_cpus`** (*`int`*) – Number of CPUs to be used for the *metric* or *non-metric* MDS
- **`seed`** (*`int or None`*) – Random seed for *metric* or *non-metric* MDS, None if not required

Returns

m (pandas.DataFrame)
Population structure after MDS (n, m)

`pyseer.input.`**`load_var_block`**(*var_type*, *p*, *burden*, *burden_regions*, *infile*, *all_strains*, *sample_order*, *min_af*, *max_af*, *max_missing*, *uncompressed*, *block_size*)

Make in iterable to load blocks of variants for LMM

Parameters

- **`var_type`** (*`str`*) – Variants type (one of: kmers, vcf or Rtab)
- **`p`** (*`pandas.DataFrame`*) – Phenotype vector (n, 1)
- **`burden`** (*`bool`*) – Whether to slice a vcf file by burden regions
- **`burden_regions`** (*`collections.deque`*) – Burden regions to slice the vcf with
- **`infile`** (*`opened file`*) – Handle to opened variant file
- **`all_strains`** (*`set-like`*) – All sample labels that should be present
- **`sample_order`** – Sampes order to interpret each Rtab line
- **`min_af`** (*`float`*) – Minimum allele frequency (inclusive)
- **`max_af`** (*`float`*) – maximum allele frequency (inclusive)
- **`max_missing`** (*`float`*) – maximum missing frequency

- **uncompressed** (*bool*) – Whether the kmers file is uncompressed
- **block_size** (*int*) – How many variants to be loaded at once

Returns

variants (iterable)
A collection of pyseer.classes.LMM objects describing the loaded variants (n,)

variant_mat (numpy.array)
Variant bloack presence/absence matrix (n, block_size)

eof (bool)
Whether we are at the end of the file

pyseer.input.**open_variant_file**(*var_type*, *var_file*, *burden_file*, *burden_regions*, *uncompressed*)

Open a variant file for use as an iterable

Parameters

- **var_type** (*str*) – Type of variants file (kmers, vcf, Rtab)
- **var_file** (*str*) – Location of file
- **burden_file** (*str*) – File containing regions to group burden tests
- **burden_regions** (*list*) – List of burden regions to be filled in-place
- **uncompressed** (*bool*) – True if kmer file is not gzipped

pyseer.input.**read_variant**(*infile*, *p*, *var_type*, *burden*, *burden_regions*, *uncompressed*, *all_strains*, *sample_order*, *keep_list=None*, *noparse=False*)

Read input line and parse depending on input file type

Return a variant name and pres/abs vector

Parameters

- **infile** (*opened file*) – Handle to opened variant file
- **p** (*pandas.Series*) – Phenotypes vector (n, 1)
- **var_type** (*str*) – Variants type (one of: kmers, vcf or Rtab)
- **burden** (*bool*) – Whether to slice a vcf file by burden regions
- **burden_regions** (*collections.deque*) – Burden regions to slice the vcf with
- **uncompressed** (*bool*) – Whether the kmers file is uncompressed
- **all_strains** (*set-like*) – All sample labels that should be present
- **sample_order** – Samples order to interpret each Rtab line
- **keep_list** (*dict*) – Variant names to properly read, any other will return None

  (default = None)
- **noparse** (*bool*) – Set True to skip line without parsing and return None, irrespective of presence in skip_list

  (default = False)

Returns

eof (bool)
Whether we are at the end of the file

k (numpy.array)
Variant presence/absence vector

var_name (str)
Variant name

kstrains (list)
Samples in which the variant is present

nkstrains (list)
Samples in which the variant is absent

af (float)
Allele frequency

missing (float)
Missing frequency

`pyseer.input.`**`read_vcf_var`**(*variant*, *d*, *keep_list=None*)

Parses vcf variants from pysam

Returns None if filtered variant. Mutates passed dictionary d

Parameters

- **`variant`** (*`pysam.libcbcf.VariantRecord`*) – Variant to be parsed
- **`d`** (*`dict`*) – Dictionary to be populated in-place
- **`keep_list`** (*`list`*) – List of variants to read

1.8.2 model.py

Original SEER model (fixed effects) implementations

`pyseer.model.`**`firth_likelihood`**(*beta*, *logit*)

Convenience function to calculate likelihood of Firth regression

Parameters

- **`beta`** (*`numpy.array`*) – (n, 1)
- **`logit`** (*`statsmodels.discrete.discrete_model.Logit`*) – Logistic model

Returns

likelihood (float)
Firth likelihood

`pyseer.model.`**`fit_firth`**(*logit_model*, *start_vec*, *X*, *y*, *step_limit=1000*, *convergence_limit=0.0001*)

Do firth regression

Parameters

- **`logit`** (*`statsmodels.discrete.discrete_model.Logit`*) – Logistic model
- **`start_vec`** (*`numpy.array`*) – Pre-initialized vector to speed-up convergence (n, 1)
- **`X`** (*`numpy.array`*) – (n, m)
- **`y`** (*`numpy.array`*) – (n,)
- **`step_limit`** (*`int`*) – Maximum number of iterations
- **`convergence_limit`** (*`float`*) – Convergence tolerance

Returns

intercept (float)
Intercept

kbeta (float)
Variant beta

beta (iterable)
Covariates betas (n-2)

bse (float)
Beta std-err

fitll (float or None)
Likelihood of fit or None if could not fit

`pyseer.model.`**`fit_lineage_effect`**`(`*lin*, *c*, *k*`)`

Fits the model $k \sim Wa$ using binomial error with logit link. W are the lineages (either a projection of samples, or cluster indicators) and covariates. Returns the index of the most significant lineage

Parameters

- **`lin`** (*`numpy.array`*) – Population structure matrix or lineage association binary matrix (n, k)
- **`c`** (*`numpy.array`*) – Covariants matrix (n, j)
- **`k`** (*`numpy.array`*) – Variant presence-absence vector (n, 1)

Returns

max_lineage (int or None)
Index of the most significant lineage or None is could not fit

`pyseer.model.`**`fit_null`**`(`*p*, *m*, *cov*, *continuous*, *firth=False*`)`

Fit the null model i.e. regression without k-mer

$y \sim Wa$

Returns log-likelihood

Parameters

- **`p`** (*`numpy.array`*) – Phenotypes vector (n, 1)
- **`m`** (*`numpy.array`*) – Population structure matrix (n, k)
- **`cov`** (*`pandas.DataFrame`*) – Covariants dataframe (n, j)
- **`continous`** (*`bool`*) – Whether phenotypes are continuous or binary
- **`firth`** (*`bool`*) – For binary phenotypes whether to use firth regression

Returns

null_res (statsmodels.regression.linear_model.RegressionResultsWrapper or float or None)
Fitted model or log-likelihood (if firth) or None if could not fit

`pyseer.model.`**`fixed_effects_regression`**`(`*variant*, *p*, *k*, *m*, *c*, *af*, *pattern*, *lineage_effects*, *lin*, *pret*, *lrtt*, *null_res*, *null_firth*, *kstrains*, *nkstrains*, *continuous*`)`

Fits the model $y \sim Xb + Wa$ using either binomial error with logit link (binary traits) or Gaussian error (continuous traits)

- y is the phenotype

- *X* is the variant presence/absence (fixed effects)
- *W* are covariate fixed effects, including population structure
- *a* and *b* are slopes to be fitted

Parameters

- **variant** (*str*) – Variant identifier
- **p** (*numpy.array*) – Phenotype vector (binary or continuous) (n, 1)
- **k** (*numpy.array*) – Variant presence/absence vector (n, 1)
- **m** (*numpy.array*) – Population structure matrix (n, m)
- **c** (*numpy.array*) – Covariants matrix (n, j)
- **af** (*float*) – Allele frequency
- **pattern** (*str*) – Variant hashed pattern
- **lineage_effects** (*bool*) – Whether to fit lineages or not
- **lin** (*numpy.array*) – Lineages matrix (n, k)
- **pret** (*float*) – Pre-filtering p-value threshold
- **lrtt** (*float*) – Post-fitting p-value threshold
- **null_res** (*float or statsmodels.regression.linear_model.RegressionResultsWrapper*) – Null-fit likelihood (binary) or model (continuous)
- **null_firth** (*float*) – Firth regression likelihood
- **kstrains** (*iterable*) – Sample labels with the variant
- **nkstrains** (*iterable*) – Sample labels without the variant
- **continuous** (*bool*) – Whether the phenotype is continuous or not

Returns

result (pyseer.classes.Seer)
Results container

pyseer.model.**pre_filtering**(*p*, *k*, *continuous*)

Calculate a naive p-value from a chisq test (binary phenotype) or a t-test (continuous phenotype) which is not adjusted for population structure

Parameters

- **p** (*numpy.array*) – Phenotypes vector (n, 1)
- **k** (*numpy.array*) – Variant presence-absence vector (n, 1)
- **continous** (*bool*) – Whether phenotypes are continuous or binary

Returns

prep (float)
Naive p-value

bad_chisq (boolean)
Whether the chisq test had small values in the contingency table

1.8.3 lmm.py

LMM interface implementations

`pyseer.lmm.fit_lmm`(*lmm*, *h2*, *variants*, *variant_mat*, *lineage_effects*, *lineage_clusters*, *covariates*, *continuous*, *filter_pvalue*, *lrt_pvalue*)

Fits LMM and returns LMM tuples for printing

Parameters

- **lmm** (*pyseer.fastlmm.lmm_cov.LMM*) – Initialised LMM model
- **h2** (*float*) – Trait's variance explained by covariates
- **variants** (*iterable*) – Tuples with variant object, phenotype vector and variant vector (pyseer.classes.LMM, numpy.array, numpy.array)
- **variant_mat** (*numpy.array*) – Variants presence absence matrix (n, k)
- **lineage_effects** (*bool*) – Whether to fit lineage effects
- **lineage_clusters** (*numpy.array*) – Population structure matrix or lineage association binary matrix (n, k)
- **covariates** (*numpy.array*) – Covariates matrix (n, m)
- **continuous** (*bool*) – Whether the phenotype is continuous
- **filter_pvalue** (*float*) – Pre-filtering p-value threshold
- **lrt_pvalue** (*float*) – Post-fitting p-value threshold

Returns

all_variants (iterable)
All variant objects fitted or filtered

`pyseer.lmm.fit_lmm_block`(*lmm*, *h2*, *variant_block*)

Actually fits the LMM to numpy variant array see map/reduce section of _internal_single in fastlmm.association.single_snp

Parameters

- **lmm** (*pyseer.fastlmm.lmm_cov.LMM*) – Initialised LMM model
- **h2** (*float*) – Trait's variance explained by covariates
- **variant_block** (*numpy.array*) – Variants presence absence matrix (n, k)

Returns

lmm_results (dict)
LMM results for this variants block

`pyseer.lmm.initialise_lmm`(*p*, *cov*, *K_in*, *lmm_cache_in=None*, *lmm_cache_out=None*, *lineage_samples=None*)

Initialises LMM using the similarity matrix see _internal_single in fastlmm.association.single_snp

Parameters

- **p** (*pandas.Series*) – Phenotypes vector (n, 1)
- **cov** (*pandas.DataFrame*) – Covariance matrix (n, m)
- **K_in** (*str*) – Similarity matrix filename

- **lmm_chache_in** (*str or None*) – Filename for an input LMM cache, None if it has to be computed
- **lmm_chache_out** (*str or None*) – Filename to save the LMM cache, None otherwise.
- **lineage_samples** (*list or None*) – Sample names used for lineage (must match K_in)

Returns

p (pandas.Series)
Phenotype vector with the samples present in the similarity matrix

lmm (pyseer.fastlmm.lmm_cov.LMM)
Initialised LMM model

h2 (float)
Trait's variance explained by covariates

1.8.4 utils.py

Utilities

pyseer.utils.**format_output**(*item*, *lineage_dict=None*, *model='seer'*, *print_samples=False*)

Format results for a variant for stdout printing

Parameters

- **item** (*pyseer.classes.Seer or pyseer.classes.LMM*) – Variant results container
- **lineage_dict** (*list*) – Lineage labels
- **model** (*str*) – The model used
- **print_samples** (*bool*) – Whether to add the samples list to the putput

Returns

out (str)
Tab-delimited string to be printed

pyseer.utils.**set_env**(***environ*)

Temporarily set the process environment variables.

```
>>> with set_env(PLUGINS_DIR=u'test/plugins'):
...   "PLUGINS_DIR" in os.environ
True
```

```
>>> "PLUGINS_DIR" in os.environ
False
```

1.8.5 cmdscale.py

Function to perform classical MDS

`pyseer.cmdscale.cmdscale(D)`

Classical multidimensional scaling (MDS)

Parameters

`D` (*numpy.array*) – Symmetric distance matrix (n, n)

Returns

Y (numpy.array)

Configuration matrix (n, p). Each column represents a dimension. Only the p dimensions corresponding to positive eigenvalues of B are returned. Note that each dimension is only determined up to an overall sign, corresponding to a reflection.

e (numpy.array)

Eigenvalues of B (n, 1)

1.8.6 enet.py

Elastic net model implementations

`pyseer.enet.correlation_filter(p, all_vars, quantile_filter=0.25)`

Calculates correlations between phenotype and variants, giving those that are above the specified quantile

Parameters

- **p** (*pandas.DataFrame*) – Phenotype vector (n, 1)
- **all_vars** (*scipy.sparse.csr_matrix*) – Narrow sparse matrix representation of all variants to fit to (rows = variants, columns = samples)
- **quantile_filter** (*float*) – The quantile to discard at e.g. 0.25, retain top 75%

  [default = 0.25]

Returns

cor_filter (numpy.array)

The indices of variants passing the filter

`pyseer.enet.enet_predict(enet_fit, variants, continuous, responses=None)`

Use a fitted elastic net model to make predictions about new observations. Returns accuracy if true responses known

Parameters

- **enet_fit** (*cvglmnet*) – An elastic net model fitted using cvglmnet or similar
- **variants** (*scipy.sparse.csc_matrix*) – Wide sparse matrix representation of all variants to predict with (rows = samples, columns = variants)
- **continuous** (*bool*) – True if a continuous phenotype, False if a binary phenotype
- **responses** (*np.array*) – True phenotypes to calculate R^2 with

  [default = None]

Returns

preds (numpy.array)
Predicted phenotype for each input sample in variants

R2 (float)
Variance explained by model (or None if true labels not provided).

`pyseer.enet.`**`find_enet_selected`**(*enet_betas*, *var_indices*, *p*, *c*, *var_type*, *fit_seer*, *burden*, *burden_regions*, *infile*, *all_strains*, *sample_order*, *continuous*, *find_lineage*, *lin*, *uncompressed*)

Read through the variant input file again, yielding just those variants which had a non-zero slope for printing

Parameters

- **enet_betas** (*numpy.array*) – Fitted slopes of intercept, covariants and variants from elastic net
- **var_indices** (*list*) – The 0-indexed locations (in the original file) of variants represented in enet_betas
- **p** (*pandas.DataFrame*) – Phenotype vector (n, 1)
- **c** (*numpy.array*) – Covariate matrix (n, j)
- **var_type** (*str*) – Variants type (one of: kmers, vcf or Rtab)
- **(tuple** (*fit_seer*) – m, null_model, null_firth) Distance projection and null models required to fit fixed effect regression
- **burden** (*bool*) – Whether to slice a vcf file by burden regions
- **burden_regions** (*collections.deque*) – Burden regions to slice the vcf with
- **infile** (*opened file*) – Handle to opened variant file
- **all_strains** (*set-like*) – All sample labels that should be present
- **sample_order** – Sample order to interpret each Rtab line
- **continuous** (*bool*) – Is phenotype/fit continuous?
- **lineage_effects** (*bool*) – Whether to fit lineages or not
- **lin** (*numpy.array*) – Lineages matrix (n, k)
- **uncompressed** (*bool*) – Whether the kmers file is uncompressed

Returns

variant (var_obj.Enet)
Iterable of processed variants for printing

`pyseer.enet.`**`fit_enet`**(*p*, *variants*, *covariates*, *weights*, *continuous*, *alpha*, *lineage_dict=None*, *fold_ids=None*, *n_folds=10*, *n_cpus=1*)

Fit an elastic net model to a set of variants. Prints information about model fit and prediction quality to STDERR

Parameters

- **p** (*pandas.DataFrame*) – Phenotype vector (n, 1)
- **variants** (*scipy.sparse.csc_matrix*) – Wide sparse matrix representation of all variants to fit to (rows = samples, columns = variants)
- **covariates** (*pandas.DataFrame*) – Covariate matrix (n, j)
- **weights** (*np.array*) – Vector of sample weights (n, 1)
- **continuous** (*bool*) – If True fit a Gaussian error model, otherwise Bionomial error

- `alpha` (*float*) – Between 0-1, sets the mix between ridge regression and lasso regression
- `lineage_dict` (*list*) – Names of lineages, indices corrsponding to fold_ids

  [default = None]
- `fold_ids` (*list*) – Index of fold assignment for cross-validation, from 0 to 1-n_folds

  [default = None]
- `n_folds` (*int*) – Number of folds in cross-validation

  [default = 10]
- `n_cpus` (*int*) – Number of processes to use in cross-validation Set to -1 to use all available

  [default = 1]

Returns

betas (numpy.array)
The fitted betas (slopes) for each variant

`pyseer.enet.load_all_vars`(*var_type*, *p*, *burden*, *burden_regions*, *infile*, *all_strains*, *sample_order*, *min_af*, *max_af*, *max_missing*, *uncompressed*)

Load all variants in the input file into a sparse matrix representation

Parameters

- `var_type` (*str*) – Variants type (one of: kmers, vcf or Rtab)
- `p` (*pandas.DataFrame*) – Phenotype vector (n, 1)
- `burden` (*bool*) – Whether to slice a vcf file by burden regions
- `burden_regions` (*collections.deque*) – Burden regions to slice the vcf with
- `infile` (*opened file*) – Handle to opened variant file
- `all_strains` (*set-like*) – All sample labels that should be present
- `sample_order` – Sampes order to interpret each Rtab line
- `min_af` (*float*) – Minimum allele frequency (inclusive)
- `max_af` (*float*) – maximum allele frequency (inclusive)
- `max_missing` (*float*) – maximum missing frequency
- `uncompressed` (*bool*) – Whether the kmers file is uncompressed

Returns

variants (scipy.sparse.csr_matrix)
A sparse matrix representation of all variants in the input

selected_vars (list)
0-Indices of variants in the input file in variants (which passed AF filtering)

var_idx (int)
The number of read variants (number of rows of variants)

`pyseer.enet.write_lineage_predictions`(*true_values*, *predictions*, *fold_ids*, *lineage_dict*, *continuous*, *stderr_print=True*)

Writes prediction ability stratified by lineage to stderr

Parameters

- **true_values** (*np.array*) – Observed values of phenotype
- **predictions** (*np.array*) – Predicted phenotype values
- **lineage_dict** (*list*) – Names of lineages, indices corrsponding to fold_ids
- **fold_ids** (*list*) – Index of fold assignment for cross-validation, from 0 to 1-n_folds
- **continuous** (*bool*) – True if a continuous phenotype, False if a binary phenotype
- **stderr_print** (*bool*) – Print output to stderr

  [default = True]

Returns

R2_vals (list)
R2 values for each fold

confusion (list)
Tuple of tn, fp, fn, tp for each fold

CHAPTER

TWO

INDEX:

- genindex
- search

PYTHON MODULE INDEX

INDEX