28/01/2020

Manual for *k*-mers-based GWAS

This manual details how to build a *k*-mers presence/absence table and how to use it to run *k*-mers-based GWAS.

The **first part** lists 4 steps to build a *k*-mers presence/absence table from raw sequencing reads. The **second part** lists other functionalities in this library. The **third part** explains how to run *k*-mers-based GWAS. Lastly, in the **fourth part** we added two general notes.

(I) Building *k*-mers presence/absence table from raw sequencing reads:

First you need to decide what is the *k*-mer length to use. The length can not exceed 31 in the current implementation, and should be the same for all individuals. We usually used a *k*-mer length of 31.

Step 1: download the *k*-mers GWAS package

Download and unzip the *k*-mers GWAS library (release v0.2-beta):

```
user@server:/$ wget
https://github.com/voichek/kmersGWAS/releases/download/v0.2-beta
/v0_2_beta.zip
user@server:/$ unzip v0.2-beta.zip
```

Prerequisites:

- Linux system with a 64 bit CPU
- R
- python**2.7**
- KMC (part of the release under external_programs/ directory)
- GEMMA (part of the release under external_programs/ directory)
- R packages (if not present the pipeline will try to automatically install):
    - MASS
    - Mvnpermute
    - matrixcalc

Step 2: Count *k*-mers and sort for each individual separately

This step will be run separately for each individual/sample. We will combine the output of all individuals in the next steps.

Create a directory for the individual:

```
user@server:kmers_proj$ mkdir individualX
user@server:kmers_proj$ cd individualX
```

Create a textual file within the individual folder with the list of sequencing files, a path for each file in a new line. For example:

```
user@server:individualX$ cat input_files.txt
/server/data/my_project/individualX_run1/seq_R1.fastq.gz
/server/data/my_project/individualX_run1/seq_R2.fastq.gz
/server/data/my_project/individualX_run3/seq_R1.fastq.gz
```

Note: as we count *k*-mers the coupling of paired-end sequencing files are not needed.

Run KMC for the first time, with canonized count[1] of *k*-mers:

```
user@server:individualX$ kmc -t2 -k31 -ci2 @input_files.txt
output_kmc_canon ./ 1> kmc_canon.1 2> kmc_canon.2
```

Parameters used (more information in KMC's manual):

-t - Number of threads to use

-k - *k*-mer length (should be between 15-31)

-ci - Threshold for counted *k*-mers. For example in the above example *k*-mer has to appear at least 2 times to be counted. This parameter depends on your coverage, but should be the same for all the individuals.

Run KMC for the second time, counting all *k*-mers with no canonization:

```
user@server:individualX$ kmc -t2 -k31 -ci0 -b @input_files.txt
output_kmc_all ./ 1> kmc_all.1 2> kmc_all.2
```

[1] See note in part 4 regarding canonization

```
Parameters used (more information in KMC's manual):
-b   - do not canonized k-mers
-ci  - this time we count all k-mers. In this step ci has to be
set to 0.
```

Combine the information from the two KMC runs to one list of *k*-mers:

```
user@server:individualX$
<KMERS-GWAS-PATH>/bin/kmers_add_strand_information -c
output_kmc_canon -n output_kmc_all -k 31 -o kmers_with_strand
```

```
Parameters used:
-c   - prefix of KMC DB files, run with canonization of k-mers
-n   - prefix of KMC DB files, run without canonization of
k-mers
-k   - k-mer length
-o   - output file name
```

This process should be repeated for each individual we wish to use in our GWAS analysis.
The KMC files, especially from the second run, can be large. Therefore, it is recommended to remove the KMC files once the information was combined[2]:

```
user@server:individualX$ rm *.kmc*
```

Useful statistics to collect: collect in one table for all individuals: (i) the number of reads, (ii) the number of unique canonized *k*-mers, (iii) the number of non-canonized *k*-mers, and (iv) the number of *k*-mers with each type of flag (1,2, or 3). These numbers are found in the log files created by the KMC's runs or by the log from *kmers_add_strand_information*. Make sure the numbers make sense. For example, plot the number of unique *k*-mers as a function of the number of reads. You expect to see a correlation between the two, with number of canonized *k*-mers showing signs of saturation (weaker slope for higher values). It is also important to notice if there are individuals which deviate significantly from the general trend, which can be due to technical issues. These specific samples are the ones you might want to omit from further analysis.

Step 3: List all *k*-mers to be used in GWAS from all individuals

In this step we will define the set of *k*-mers used in the *k*-mers table. We will go over the *k*-mers lists from each individual and combine them to one list. The *k*-mers will be filtered according to two criteria: First, a *k*-mer needs to appear in at least X (e.g. X=5) individuals. Second, a *k*-mer have to appear in each canonized/non-canonized form in at least *P* percent (e.g. *P*=20%) of individuals from which it appeared in.

[2] You can save the statistics of *k*-mers appearances (see second part) before deleting the KMC files

For example: X=5, P=20%, and we have 1000 individuals

- *k*-mer **x** was found in 4 individuals - **x** is filtered as 4<X.
- *k*-mer **x** was found in 98 individuals: 10 individuals in canonized and non-canonized form, in 7 only in canonized form, and in 81 only in non-canonized form - **x** will be filtered as it appeared only in 17 (7+10) individuals in non-canonized form. However, 17 < 98*0.20=19.6.

We will create a list of all individuals *k*-mers list files. Each line in the file will have the full path of the file with the *k*-mers list, followed by a **<tab>** and the individual name. For example:

```
user@server:kmers_proj$ cat kmers_list_paths.txt
/server/kmers_proj/individualX/kmers_with_strand  X
/server/kmers_proj/individualY/kmers_with_strand  Y
/server/kmers_proj/individualZ/kmers_with_strand  Z
/server/kmers_proj/individualW/kmers_with_strand  W
…
```

For example, you can use the following command to create this file (if you want to include all subdirectories):

```
ll /server/kmers_proj/ | tail -n +2 | awk '{printf
"/server/kmers_proj/%s/kmers_with_strand\t%s\n", $NF,$NF}' >
kmers_list_paths.txt
```

Combine and filter lists of *k*-mers to one file:

```
user@server:kmers_proj$
<KMERS-GWAS-PATH>/bin/list_kmers_found_in_multiple_samples -l
kmers_list_paths.txt -k 31 --mac 5 -p 0.2 -o kmers_to_use
```

Parameters used:

-l - list of separate *k*-mers files
-k - length of *k*-mers
--mac - minor allele count (min allowed appearance of a *k*-mer)
-p - minimum percent of appearance in each strand form
-o - output file name

Files outputted:

1. kmers_to_use - binary file with the filtered list of *k*-mers to use.
2. kmers_to_use.shareness - textual file with counts of how many *k*-mers appeared in any number of individuals. Only take into account *k*-mers which were not filtered out.

3. kmers_to_use.stats.both / kmers_to_use.stats.only_canonical / kmers_to_use.stats.only_non_canonical - textual files containing a matrix of size (N+1)x(N+1), where N is the number of individuals. In row **i** and column **j**, the number indicates the number of *k*-mers which appeared (i-1) times in total and (j-1) times in the files specific form, that is in both forms, only in canonical form, or only in non-canonical form, respectively.

Step 4: Create the *k*-mers table

Now we can create the *k*-mers table, containing the presence/absence pattern of each *k*-mer in the population.

```
user@server:kmers_proj$ <KMERS-GWAS-PATH>/bin/build_kmers_table
-l kmers_list_paths.txt -k 31 -a kmers_to_use -o kmers_table
```

```
Parameters used:
-l   - list of separate k-mers files
-k   - length of k-mers
-a   - path to file with all k-mers
-o   - prefix for k-mers table files
```

Files outputted:

1. kmers_table.table - binary file with all the *k*-mers presence/absence information.
2. kmers_table.names - textual file with name of all individual in seperate lines.

After creating the *k*-mers table, there is no longer a need for separate *k*-mers list from each individual. Therefore, to save space, all these files (...*kmers_with_strand*) can be removed.

(II) Other functionalities of the library

Calculation of kinship matrix

To calculate the kinship matrix from the *k*-mers table run the following:

```
user@server:kmers_proj$ <KMERS-GWAS-PATH>/bin/emma_kinship_kmers
-t kmers_table -k 31 --maf 0.05 > kmers_table.kinship
```

Parameters used:
-t - *k*-mers table prefix path
-k - length of *k*-mers
--maf - minor allele frequency

Files outputted:
kmers_table.kinship - a textual file with a matrix of size NxN, where N is the number of individuals. In position i,j in the matrix there is the relationship between individual i and j. Relatedness are calculated as in EMMA[3], where 0 is the minimum relatedness and 1 is the maximal. The order of the samples is the same as in the kmers_list_paths.txt file.

Note on performance: This command might take a few days to run, according to the number of variants included in the analysis (the MAF threshold affects the running time). For example, for MAF of 0.01 on ~1000 *A. thaliana* individuals it took around 5 days to run. This command can be significantly optimized if needed, however, as it has to run only once for a project, we see no reason to do so for now.

k-mers table conversion to PLINK binary format

The *k*-mers table can be converted to PLINK binary file, coding *k*-mers presence/absence as two homozygous variants. This option is useful if you want to run GWAS on all *k*-mers using an external GWAS software, and it can be used for other purposes.
To convert the *k*-mers table, run the following:

```
user@server:kmers_proj$ <KMERS-GWAS-PATH>/bin/kmers_table_to_bed
-t kmers_table -k 31 -p phenotypes.pheno --maf 0.05 --mac 5 -b
10000000 -o output_file
```

This procedure condenses the *k*-mers table only to the individuals specified in the *phenotypes.pheno* file. It will also filter for the minor allele count (MAC) and minor allele frequency (MAF) provided by the user. The output matrix can be separated into smaller matrices of size smaller than the defined maximum batch size. This is done so one can run association mapping in parallel on each sub-matrix.

[3] https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3386377/

```
Parameters used:
-t    - k-mers table prefix path
-k    - length of k-mers
-p    - phenotype file, condense output only to individuals with
a phenotype
--maf      - minor allele frequency
--mac      - minor allele count
-b    - maximal number of variants in each PLINK bed file
      (separate to many files)
-o    - prefix for output files
-u    - output only unique presence/absence patterns (optional)
```

Format of phenotype file: the phenotype file should be with two columns separated by tabs (\t), with “accession_id” and “phenotype_value” in the header. The first column lists the name of the individuals, as in the kmers_table, and the second, phenotype_value, the phenotypic values in numerical form.

This PLINK binary files can then be used as input to SNP-based GWAS program. Notice that in this case you will also need to condense the kinship matrix (*k*-mers- or SNP- based) to the same individuals in the phenotype file, and in the same order.

Textual output of *k*-mers presence/absence pattern

Output the presence/absence pattern of a list of *k*-mers in textual format:

```
user@server:kmers_proj$ <KMERS-GWAS-PATH>/bin/filter_kmers -t
kmers_table -k kmers_list.txt -o output.txt
```

```
Parameters used:
-t    - k-mers table prefix path
-k    - file with k-mers, each k-mer in a seperate line
-o    - output file
```

Run *k*-mers-based GWAS with permutation based-threshold

Basic usage:

To run *k*-mers based GWAS on your phenotype you need to run the python wrapper script. For example:

```
user@server:kmers_proj$ python2.7
<KMERS-GWAS-PATH>/kmers_gwas.py --pheno phenotype.pheno
--kmers_table kmers_table -l 31 -p 8 --outdir output_dir
```

```
Parameters used:
--pheno    - a file with numerical phenotypic information (format
described in "k-mers table conversion to PLINK binary format"
section)
--kmers_table  - path to k-mers table
-l    - k-mers length
-p    - maximum number of threads to use
--outdir  - path to a directory for output
```

Output files inside the output directory:

The output directory contains many intermediate files of the pipeline. The following files are the most relevant for the user:

- kmers/pass_threshold_5per - the GEMMA results passing the 5% threshold
- kmers/pass_threshold_10per - the GEMMA results passing the 10% threshold
- kmers/output/phenotype_value.assoc.txt - GEMMA results for all *k*-mers passing the first filtering of the pipeline
- kmers/output/phenotype_value.assoc.txt.gz - all the GEMMA results
- kmers/threshold_5per - the (-log10) threshold for 5% family-wise error rate
- kmers/threshold_10per - the (-log10) threshold for 10% family-wise error rate
- kmers/pheno.pattern_counter - the number of unique presence/absence patterns of *k*-mers used in the GWAS (only if “--pattern_counter” was used)

The *k*-mers names contain a number (e.g "AT...GAC_**145**") the number is the position of the *k*-mer when ordering the *k*-mers in the first step of the pipeline.

Advanced options:

```
-k, --kmers_number  - number of k-mers to filter from first step
--kmers_for_no_perm_phenotype - number of k-mers to filter from
first step only for the non-permuted phenotype (if different)
--permutations - number of permutations for defining the
threshold (should be at least 20)
--maf     - minor allele frequency
--mac     - minor allele count
--min_data_points   - stop running with less phenotypic
individuals than this threshold
--pattern_counter   - count the number of unique
presence/absence k-mers patterns
--dont_remove_intermediates   - keep the intermediate files
--gemma_path   - path to GEMMA (tested with version 0.96)
```

Compare to SNP-based GWAS

This pipeline was initially built to compare the *k*-mers-based GWAS to the SNPs-based one. Therefore, there are also options to run the pipeline on both SNPs and *k*-mers, so it will use the exact same permutations and all other parameters will be equal.

```
--run_on_snps_one_step   - run pipeline with the same parameters
on SNPs (for comparison)
--run_on_snps_two_steps  - run pipeline with the same parameters
on SNPs - first approximate for initial filtering and then using
GEMMA on top hits
--snp_matrix   - base name for snps bed/bim/fam files
--snps_number  - numbers of snps to filter from the first step
(used when running a two step snps approximation)
--dont_run_on_kmers - don't run pipeline on k-mers
--kinship_snps - use the kinship matrix from the SNPs table
(should be in the SNP matrix base name .kinship)
```

Output for SNP-based analysis will be under the **snps** directory, with the same files described above for *k*-mers.

A few general notes

k-mers canonization

Canonization of *k*-mers just mean that the *k*-mer (e.g. AGGCT) and its reverse-complement (e.g. AGCCT) are the same for the purpose of counting or presence/absence patterns. We count *k*-mers twice per sample so we will have in addition to the *k*-mers which appear more than X times, also which form they appeared in.

We add this extra layer of information so we can filter out (if we want) *k*-mers which tend to appear only in one form and not the other. This can be, for example, due to sequencing of adaptors.

k-mers and RAD-Seq

We had only one experience using this method with reduced sequence representation by RAD-Seq. We found, at least in that example, that some modifications should be applied. First, in our case the protocol was strand specific and therefore filtering *k*-mers appearing in only one strand will be ill-advised (this can be achieved by setting -p 0 in step 3). Second, we observed that this data is more sensitive to coverage differences, and therefore down-sampled our samples to have the same coverage. We reasoned that this is due to the size selection applied in this protocol, and hence higher coverage will sample more from the tails of the fragment size distribution.

We optimized these steps by testing different parameters and looking on the kinship matrix, which had an expected pattern.

Finding the reads a *k*-mer originated from

Once we found a *k*-mer that is associated with the phenotype we might want to find the short sequence reads it originated from. To find the reads containing a *k*-mer of interest you have to do two things. First, you need to find in which accessions/strains the *k*-mer was present. This can be done using the **filter_kmers** functionality. Then you need to filter the reads and look for reads from the sample of interest and filter the ones that contain your *k*-mer(s). You can do this by writing your own script, I wrote a small program to do that but as I have seen there are specialized programs (e.g. RiDiCulousFilter) that do just that, I didn't find a reason to upload mine.