Package ‘skater’

October 14, 2022

Title Utilities for SNP-Based Kinship Analysis

Description Utilities for single nucleotide polymorphism (SNP) based kinship analysis

testing and evaluation. The 'skater' package contains functions for importing, parsing,

and analyzing pedigree data, performing relationship degree inference, benchmarking

relationship degree classification, and summarizing identity by descent (IBD) segment data.

Package functions and methods are described in Turner et al. (2021) "skater: An R package


Version 0.1.1

License MIT + file LICENSE

URL https://github.com/signaturescience/skater

BugReports https://github.com/signaturescience/skater/issues

Encoding UTF-8

RoxygenNote 7.1.2

Depends R (>= 3.0.0)

Imports magrittr, dplyr, tidyr, readr, purrr, kinship2, corrr,
grDevices, tibble, rlang

Suggests rmarkdown, markdown, knitr, testthat (>= 3.0.0)

Config/testthat/edition 3

VignetteBuilder knitr

NeedsCompilation no

Author Stephen Turner [aut, cre] (<https://orcid.org/0000-0001-9140-9028>),

Matthew Scholz [aut] (<https://orcid.org/0000-0003-3686-1227>),

VP Nagraj [aut] (<https://orcid.org/0000-0003-0060-566X>),

Signature Science, LLC. [cph]

Maintainer Stephen Turner <vustephen@gmail.com>

Repository CRAN

Date/Publication 2022-02-01 16:00:02 UTC
Description

Some types of data or results are indexed by two identifiers in two different columns corresponding to data points for *pairs* of observations. E.g., you may have columns called `id1` and `id2` that index the tibble for all possible pairs of results between samples A, B, and C. If you attempt to join two tibbles with `by=c("id1", "id2")`, the join will fail if samples are flipped from one dataset to another. E.g., one tibble may have `id1=A` and `id2=B` while the other has `id1=B` and `id2=A`. This function ensures that `id1` is alphanumerically first while `id2` is alphanumerically second. See examples.

Usage

```
arrange_ids(.data, .id1, .id2)
```

Arguments

- `.data` A tibble with two ID columns to arrange.
- `.id1` Unquoted name of the "id1" column. See examples.
- `.id2` Unquoted name of the "id2" column. See examples.
Value

A tibble with id1 and id2 rearranged alphanumerically.

Examples

d1 <- tibble::tribble(~id1, ~id2, ~results1,
  "a", "b", 10L,
  "a", "c", 20L,
  "c", "b", 30L)
d2 <- tibble::tribble(~id1, ~id2, ~results2,
  "b", "a", 101L,
  "c", "a", 201L,
  "b", "c", 301L)

# Inner join fails because id1!=id2.
dplyr::inner_join(d1, d2, by=c("id1", "id2"))

# Arrange IDs
d1 %>% arrange_ids(id1, id2)
d2 %>% arrange_ids(id1, id2)

# Inner join
dplyr::inner_join(arrange_ids(d1, id1, id2), arrange_ids(d2, id1, id2), by=c("id1", "id2"))

# Recursively, if you had more than two tibbles
list(d1, d2) %>%
purrr::map(arrange_ids, id1, id2) %>%
purrr::reduce(dplyr::inner_join, by=c("id1", "id2"))

---

calc_accuracy  Calculate Accuracy

description

Calculates accuracy and related metrics.

Usage

calc_accuracy(tabble)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tabble</td>
<td>A frequency table created with <code>table</code></td>
</tr>
</tbody>
</table>

Details

Calculates accuracy, lower and upper bounds, the guessing rate and p-value of the accuracy vs. the guessing rate. This function is called by `confusion_matrix`, but if this is all you want, you can simply supply the table to this function.
calc_stats

Value
A tibble with the corresponding statistics

Author(s)
Michael Clark (see m-clark/confusion_matrix).

See Also
binom.test

calc_stats  Calculate various statistics from a confusion matrix

Description
Given a frequency table of predictions versus target values, calculate numerous statistics of interest.

Usage
calc_stats(tabble, prevalence = NULL, positive, ...)

Arguments
tabble  A frequency table created with table
prevalence  Prevalence value. Default is NULL
positive  Positive class
...  Other, not currently used

Details
Used within confusion_matrix to calculate various confusion matrix metrics. This is called by confusion_matrix, but if this is all you want you can simply supply the table.

Suppose a 2x2 table with notation

<table>
<thead>
<tr>
<th></th>
<th>Predicted</th>
<th>target</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event</td>
<td></td>
<td>Event</td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>No Event</td>
<td>C</td>
<td>D</td>
</tr>
</tbody>
</table>

The formulas used here are:

\[
\text{Sensitivity} = \frac{A}{A + C}
\]
\[
\text{Specificity} = \frac{D}{B + D}
\]
\[
\text{Prevalence} = \frac{A + C}{A + B + C + D}
\]
PositivePredictiveValue = (sensitivity*prevalence)/((sensitivity*prevalence)+((1−specificity)*(1−prevalence)))
NegativePredictiveValue = (specificity*(1−prevalence))/(((1−sensitivity)*prevalence)+((specificity)*(1−prevalence)))

DetectionRate = A/(A + B + C + D)
DetectionPrevalence = (A + B)/(A + B + C + D)
BalancedAccuracy = (sensitivity + specificity)/2

Precision = A/(A + B)
Recall = A/(A + C)

F1 = harmonicmeanofprecisionandrecall = (1+beta^2)*precision*recall/((beta^2*precision)+recall)

where beta = 1 for this function.

FalseDiscoveryRate = 1 − PositivePredictiveValue
FalseOmissionRate = 1 − NegativePredictiveValue
FalsePositiveRate = 1 − Specificity
FalseNegativeRate = 1 − Sensitivity
D' = qnorm(Sensitivity) − qnorm(1−Specificity)

AUC = pnorm(D'/sqrt(2))

See the references for discussions of the first five formulas. Abbreviations:

Positive Predictive Value: PPV
Negative Predictive Value: NPV
False Discovery Rate: FDR
False Omission Rate: FOR
False Positive Rate: FPR
False Negative Rate: FNR

Value

A tibble with (at present) columns for sensitivity, specificity, PPV, NPV, F1 score, detection rate, detection prevalence, balanced accuracy, FDR, FOR, FPR, FNR. For more than 2 classes, these statistics are provided for each class.

Note

Different names are used for the same statistics.

Sensitivity: True Positive Rate, Recall, Hit Rate, Power
Specificity: True Negative Rate
Positive Predictive Value: Precision
False Negative Rate: Miss Rate, Type II error rate, beta
False Positive Rate: Fallout, Type I error rate, alpha

This function is called by confusion_matrix, but if this is all you want, you can simply supply the table to this function.
Author(s)
Michael Clark (see m-clark/confusion_matrix).

References

---

**confusion_matrix**

*Calculate various statistics from a confusion matrix*

**Description**

Given a vector of predictions and target values, calculate numerous statistics of interest. Modified from m-clark/confusion_matrix.

**Usage**

```r
confusion_matrix(
  prediction,
  target,
  positive = NULL,
  prevalence = NULL,
  dnn = c("Predicted", "Target"),
  longer = FALSE,
  ...
)
```

**Arguments**

- `prediction`: A vector of predictions
- `target`: A vector of target values
- `positive`: The positive class for a 2-class setting. Default is NULL, which will result in using the first level of `target`.
- `prevalence`: Prevalence rate. Default is NULL.
- `dnn`: The row and column headers for the contingency table returned. Default is 'Predicted' for rows and 'Target' for columns.
- `longer`: Transpose the output to long form. Default is FALSE (requires tidyr 1.0).
- `...`: Other parameters, not currently used.
Details

This returns accuracy, agreement, and other statistics. See the functions below to find out more. Originally inspired by the `confusionMatrix` function from the `caret` package.

Value

A list of tibble(s) with the associated statistics and possibly the frequency table as list column of the first element. If classes contain >1 numeric class and a single non-numeric class (e.g., "1", "2", "3", and "Unrelated", the RMSE of the reciprocal of the Targets + 0.5 will also be returned.)

References


See Also

calc_accuracy calc_stats

Examples

```r
prediction = c(0,1,0,1,0,1,0,1,1,1)
target = c(0,1,1,0,1,0,1,0,1,1)
confusion_matrix(prediction, target, positive = '1')

set.seed(42)
prediction = sample(letters[1:4], 250, replace = TRUE, prob = 1:4)
target = sample(letters[1:4], 250, replace = TRUE, prob = 1:4)
confusion_matrix(prediction, target)

prediction = c(rep(1, 50), rep(2, 40), rep(3, 60))
target = c(rep(1, 50), rep(2, 50), rep(3, 50))
confusion_matrix(prediction, target)

# Prediction with an unrelated class
prediction = c(rep(1, 50), rep(2, 40), rep(3, 60), rep("Unrelated", 55))
target = c(rep(1, 50), rep(2, 50), rep(3, 55), rep("Unrelated", 50))
confusion_matrix(prediction, target)

# Prediction with two unrelated classes
prediction = c(rep(1, 50), rep(2, 40), rep("Third", 60), rep("Unrelated", 55))
target = c(rep(1, 50), rep(2, 50), rep("Third", 55), rep("Unrelated", 50))
confusion_matrix(prediction, target)
```

dibble

**Description**

Creates a tibble with degree, expected kinship coefficient, and inference boundaries. Rows will be created up to the max_degree, with an additional row for any relationship more distant than max_degree. The degree value for the final row will be NA. This represents inference criteria for "unrelated" individuals. See examples.

**Usage**

dibble(max_degree = 3L)

**Arguments**

max_degree  The most distant degree you want to measure (usually between 3-9, default 3).

**Value**

A tibble containing the degree, expected kinship coefficient (k), lower (l) and upper (u) inference bounds.

**Examples**

dibble(3)
dibble(10)

---

fam2ped

**Description**

Converts a PLINK-formatted fam file to a pedigree object using kinship2::pedigree.

**Usage**

fam2ped(fam)

**Arguments**

fam  A tibble with six columns of PLINK .fam data as read in by read_fam.

**Value**

A tibble with new listcol ped containing pedigrees from kinship2::pedigree.
ibd2kin

Examples

```r
famfile <- system.file("extdata", "3gens.fam", package="skater", mustWork=TRUE)
fam <- read_fam(famfile)
fam2ped(fam)
```

ibd2kin

*Compute kinship coefficient from IBD segments*

Description

This function is used to retrieve a relatedness measure from IBD segments. The relatedness value returned is the kinship coefficient.

Usage

```r
ibd2kin(.ibd_data, .map, type = NULL)
```

Arguments

- `.ibd_data`: Tibble with IBD segments created using the `read_ibd` function
- `.map`: Tibble with the genetic map data created using the `read_map` function
- `type`: Type of IBD to use for kinship coefficient calculation; must be 'IBD1', 'IBD2', or NULL (both IBD1 and IBD2 will be treated the same); default is NULL

Details

The input data should be pairwise IBD segments prepared via `read_ibd`. The function will internally loop over each chromosome, and use a specified genetic map to convert shared segments to genetic units. After doing so, the function converts the shared length to a kinship coefficient by summing $0.5 \times IBD2 + 0.25 \times IBD1$.

Note that the data read in by `read_ibd` when source="pedsim" returns a list with separate tibbles for IBD1 and IBD2 segments. The current implementation of this function requires running this function independently on IBD1 and IBD2 segments, then summarizing (adding) the corresponding proportions. See examples.

Value

Tibble with three columns:

1. id1 (sample identifier 1)
2. id2 (sample identifier 2)
3. kinship (kinship coefficient derived from shared segments)

References

http://faculty.washington.edu/sguy/ibd_relatedness.html
Examples

```r
pedsim_fp <- system.file("extdata", "GBR.sim.seg.gz", package="skater", mustWork=TRUE)
pedsim_seg <- read_ibd(pedsim_fp, source = "pedsim")
gmapfile <- system.file("extdata", "sexspec-avg-min.plink.map", package="skater", mustWork=TRUE)
gmap <- read_map(gmapfile)
ibd1_dat <- ibd2kin(.ibd_data=pedsim_seg$IBD1, .map=gmap, type="IBD1")
ibd2_dat <- ibd2kin(.ibd_data=pedsim_seg$IBD2, .map=gmap, type="IBD2")
dplyr::bind_rows(ibd1_dat, ibd2_dat) %>%
dplyr::group_by(id1, id2) %>%
dplyr::summarise(kinship = sum(kinship), .groups = "drop")
```

 interpolate

| interpolate | Interpolate over segments |

**Description**

This is an unexported helper used in in `ibd2kin`. The function interpolates over segments to apply genetic length to the segment. It is inspired by Python code distributed by the Browning lab (documentation).

**Usage**

```r
interpolate(ibd_bp, chromgpos)
```

**Arguments**

- `ibd_bp` Base pair for the IBD segment over which to interpolate
- `chromgpos` Genetic map data for a specific chromosome

**Value**

Numeric vector with the genetic distance shared at the segment.

**References**

http://faculty.washington.edu/sguy/ibd_relatedness.html
kin2cm  

**kin2cm**  

**Kinship coefficient to cM**

---

**Description**

"Converts" a kinship coefficient to put on the same scale as shared cM using the formula \( cm < -pmin(3560, 4 \times pmax(0, k) \times 3560) \).

**Usage**

kin2cm(k)

**Arguments**

- **k**  
  Kinship coefficient (numeric, typically between 0 and .5, although KING can produce values <0).

**Value**

A vector of numeric estimated cM, ranging from 0-3560.

**References**


**Examples**

```r
kin2cm(.25)
kin2cm(.125)
kin2cm(.0625)
dibble(9) %>% dplyr::mutate(cm=kin2cm(k))
```
## kin2degree

**Kinship coefficient to degree**

### Description

Infers relationship degree given a kinship coefficient.

### Usage

```
kin2degree(k, max_degree = 3L)
```

### Arguments

- **k**: Kinship coefficient (numeric, typically between 0 and .5, although KING can produce values <0).
- **max_degree**: Max degree resolution (default 3). Used to seed `dibble`. Anything below the inference range of `max_degree` will report `NA`. See `dibble`.

### Value

A vector with inferred degree, up to the maximum degree in `dibble` (anything more distant is `NA`, i.e., unrelated).

### Examples

```
kin2degree(0.5)
kin2degree(0.25)
kin2degree(0.125)
kin2degree(0.0625)
kin2degree(0.03125)
kin2degree(0.03125, max_degree=5)
kin2degree(-0.05)
k <- seq(.02, .5, .03)
kin2degree(k)
kin2degree(k, max_degree=5)
tibble::tibble(k=k) %>% dplyr::mutate(degree=kin2degree(k))
```

## ped2kinpair

**Pedigree to pairwise kinship**

### Description

Converts a pedigree class object from `fam2ped` to a pairwise list of relationships and their expected/theoretical kinship coefficient.
**plot_pedigree**

**Usage**

```
ped2kinpair(ped)
```

**Arguments**

- `ped` A "pedigree" class object from `fam2ped`.

**Value**

A tibble containing all pairwise kinship coefficients from the input pedigree.

**Examples**

```r
famfile <- system.file("extdata", "3gens.fam", package="skater", mustWork=TRUE)
famfile %>%
  read_fam() %>%
fam2ped() %>%
dplyr::mutate(kinpairs=purrr::map(ped, ped2kinpair)) %>%
dplyr::select(fid, kinpairs) %>%
tidyrr::unnest(cols=kinpairs)
```

---

**plot_pedigree**

**Plot pedigree**

**Description**

Plot pedigree

**Usage**

```
plot_pedigree(ped, file = NULL, width = 10, height = 8)
```

**Arguments**

- `ped` List of pedigree objects from `fam2ped`
- `file` Output file path (must end in ".pdf")
- `width` Width of output PDF
- `height` Height of output PDF

**Value**

No return value, called for side effects.
### read_akt

**Description**

Reads in an akt kin results file. Input file must have seven columns, whitespace delimited:

1. id1 (member 1)
2. id2 (member 2)
3. IBD0 (ratio of IBD0/All SNPS)
4. IBD1 (ratio of IBD1/All SNPS)
5. Kinship Coefficient
6. NSNPS

**Usage**

```r
read_akt(file)
```

**Arguments**

- **file**
  
  Input file path

**Value**

A tibble containing the 7 columns from the akt file.

**Examples**

```r
aktFile <- system.file("extdata", "3gens.akt", package="skater", mustWork=TRUE)
akt <- read_akt(aktFile)
akt
```

### read_fam

**Description**

Reads in a PLINK-formatted .fam file. Input file must have six columns:

1. Family ID
2. Individual ID
3. Father ID
4. Mother ID
5. Sex
6. Affected Status

```r
read_fam(file)
```

**Arguments**

- **file**
  
  Input file path

**Value**

DataFrame containing the 6 columns from the fam file.
**read_ibd**

**Usage**

`read_fam(file)`

**Arguments**

- **file**  
  Input file path

**Value**

A tibble containing the 6 columns from the fam file.

**Examples**

```r
famfile <- system.file("extdata", "3gens.fam", package="skater", mustWork=TRUE)
fam <- read_fam(famfile)
fam
```

---

**read_ibd**

*Read IBD segment file*

**Description**

Reads in the inferred IBD segments from hapibd ([documentation](#)) or IBD segment file generated by ped-sim ([documentation](#)).

If reading a hapibd segment file, the input data should have the following columns:

1. First sample identifier
2. First sample haplotype index (1 or 2)
3. Second sample identifier
4. Second sample haplotype index (1 or 2)
5. Chromosome
6. Base coordinate of first marker in segment
7. Base coordinate of last marker in segment
8. cM length of IBD segment

If read a pedsim segment file, the input data should have the following columns:

1. First sample identifier
2. Second sample identifier
3. Chromosome
4. Physical position start
5. Physical position end
6. IBD type
7. Genetic position start
8. Genetic position end
9. Genetic length (end - start)
Usage

read_ibd(file, source)

Arguments

file Input file path
source Source of the input file; must be one of "hapibd" or "pedsim"

Value

if source="hapibd", a tibble is returned. If source="pedsim", a list with two tibbles elements, IBD1 and IBD2 is returned. Both the hapibd tibble, and the two pedsim tibbles contain six columns:

1. id1 (sample identifier 1)
2. id2 (sample identifier 2)
3. chr (chromosome)
4. start (segment bp start coordinate)
5. end (segment bp end coordinate)
6. length (shared segment length in genetic units, cM)

References

https://github.com/browning-lab/hap-ibd#output-files
https://github.com/williamslab/ped-sim#output-ibd-segments-file

Examples

hapibd_fp <- system.file("extdata", "GBR.sim.ibd.gz", package="skater", mustWork=TRUE)
hapibd_seg <- read_ibd(hapibd_fp, source = "hapibd")
pedsim_fp <- system.file("extdata", "GBR.sim.seg.gz", package="skater", mustWork=TRUE)
pedsim_seg <- read_ibd(pedsim_fp, source = "pedsim")

read_ibis Read IBIS coef output file

Description

Reads in an ibis results file. Input file must have six columns, whitespace delimited:

1. id1 (member 1)
2. id2 (member 2)
3. Kinship Coefficient
4. IBD2 (ratio of IBD2/All SNPS)
5. Segment count
6. Kinship Degree
read_map

Usage
read_ibis(file)

Arguments
file  Input file path

Value
A tibble containing the 6 columns from the ibis file.

Examples
ibisFile <- system.file("extdata", "3gens.ibis.coef", package="skater", mustWork=TRUE)
ibis <- read_ibis(ibisFile)
ibis

Description
This function reads in the content from a genetic map file to translate physical distance to genetic units (i.e. cM). Regardless of the source, the input file must be sex-averaged and in a tab-separated "Plink" format (documentation) with the following four columns and no header (i.e. no column names):

1. Chromosome
2. Identifier (ignored in read_map())
3. Length (genetic length within the physical position boundary)
4. Position (physical position boundary)

The columns must be in the order above. Note that only the first, third, and fourth columns are used in the function.

Usage
read_map(file)

Arguments
file  Input file path
Details

The genetic map could come from different sources. One source is the HapMap map distributed by the Browning Lab (documentation). If this map file is used, the non-sex chromosomes can be downloaded and concatenated to a single file as follows:

```
wget http://bochet.gcc.biostat.washington.edu/beagle/genetic_maps/plink.GRCh37.map.zip
unzip plink.GRCh37.map.zip
cat *chr[0-9]*GRCh37.map | sort -k1,1 -k4,4 --numeric-sort > plink.allchr.GRCh37.map
```

Another source is a sex-specific map ("bherer") originally published by Bherer et al and recommended by the developers of ped-sim for simulating IBD segments (documentation). To retrieve and prep this map file for simulation:

```
# Get the refined genetic map and extract
tar xvfpz Refined_genetic_map_b37.tar.gz

# Format for ped-sim as per https://github.com/williamslab/ped-sim#map-file-
printf "#chr	pos	male_cM	female_cM
" > sexspec.pedsim.map
for chr in {1..22}; do
    paste Refined_genetic_map_b37/male_chr$chr.txt Refined_genetic_map_b37/female_chr$chr.txt |
        awk -v OFS="\t" '/quotesingle.Var
        NR > 1 && $2 == $6 {print $1,$2,$4,$8}' |
        sed '/quotesingle.Var
        s/^chr//'/ >> sexspec.pedsim.map;
done

# Clean up
rm -rf Refined_genetic_map_b37*
```

After this, the sexspec.pedsim.map file is ready for use in simulation. However, it must be averaged and reformatted to "Plink format" to use here:

```
cat sexspec.pedsim.map | grep -v "^#" | awk -v OFS="\t" '{print $1,".",($3+$4)/2,$2}' > sexspec-avg.plink.map
```

The genetic maps created above are in the tens of megabytes size range. This is trivial to store for most systems but a reduced version would increase portability and ease testing. This "minimum viable genetic map" could be used for testing and as installed package data in an R package for example analysis. Read more about minimum viable genetic maps at:

- Github repo with python code: https://github.com/williamslab/min_map

The code as written below reduces the averaged sex-specific genetic map from 833776 to 28726 positions (~30X reduction!).

```
# Get minmap script from github
wget https://raw.githubusercontent.com/williamslab/min_map/main/min_map.py

# Create empty minmap
echo -n > sexspec-avg-min.plink.map
```
# For each autosome...
for chr in {1..22}; do
echo "Working on chromosome $chr..."
# First pull out just one chromosome
grep "^${{chr}}[[:space:]]" sexspec-avg.plink.map > tmp.$chr
# Run the python script on that chromosome.
# The genetic map column is 3rd column (2nd in 0-start). Physical position is last column (3 in 0-based)
python3 min_map.py -mapfile tmp.$chr -chr $chr -genetcol 2 -physcol 3 -noheader -error 0.05
# Strip out the header and reformat back to plink format, and append to minmap file
cat min_viable_map${chr}.txt | grep -v "^#" | awk -v OFS="\t" '{print $1,".",$4,$2}' >> sexspec-avg-min.plink.map
# Clean up
rm -f min_viable_map${chr}.txt tmp.$chr
done

This averaged version of the Bherer sex-specific map, reduced to a minimum viable genetic map with at most 5% error, in Plink format, is available as installed package data (see examples). This is useful for testing code, but the full genetic map should be used for most analysis operations.

Value
A tibble containing 3 columns:

1. chr (chromosome)
2. value (genetic length within the physical position boundary)
3. bp (physical position boundary)

References

http://zzz.bwh.harvard.edu/plink/data.shtml#map
http://bochet.gcc.biostat.washington.edu/beagle/genetic_maps/
https://github.com/williamslab/ped-sim#map-file
https://www.nature.com/articles/ncomms14994
https://www.nature.com/articles/ncomms14994
https://github.com/cbherer/Bherer_etal_SexualDimorphismRecombination

Examples

gmapfile <- system.file("extdata", "sexspec-avg-min.plink.map", package="skater", mustWork=TRUE)
gmap <- read_map(gmapfile)
**read_plink2_king**

Read PLINK KING table

**Description**

Reads in the output from `plink2 --make-king-table` ([documentation](https://www.cog-genomics.org/plink/2.0/distance#make_king)). Input file must have six columns, tab delimited:

1. id1 (member 1)
2. id2 (member 2)
3. nsnp
4. hethet: proportion of sites where both are heterozygous
5. k: Kinship Coefficient

**Usage**

```r
read_plink2_king(file)
```

**Arguments**

- `file` Input file path

**Value**

A tibble containing the 6 columns from the `plink2 --make-king-table` output.

**References**

[https://www.cog-genomics.org/plink/2.0/distance#make_king](https://www.cog-genomics.org/plink/2.0/distance#make_king)

**Examples**

```r
plink2kingFile <- system.file("extdata", "plink2-king-table.tsv", package="skater", mustWork=TRUE)
plink2king <- read_plink2_king(plink2kingFile)
plink2king
plink2king %>% dplyr::filter(k>0.01)
```
Index

arrange_ids, 2
binom.test, 4
calc_accuracy, 3, 7
calc_stats, 4, 7
confusion_matrix, 6
dibble, 8, 12
fam2ped, 8, 12, 13
ibd2kin, 9, 10
interpolate, 10
kin2cm, 11
kin2degree, 12
kinship2::pedigree, 8
ped2kinpair, 12
plot_pedigree, 13
read_akt, 14
read_fam, 8, 14
read_ibd, 9, 15
read_ibis, 16
read_map, 9, 17
read_plink2_king, 20
table, 3, 4