# Package: FamilyRank (via r-universe)

August 22, 2024

**Type** Package

**Title** Algorithm for Ranking Predictors Using Graphical Domain Knowledge

**Version** 1.0

**Date** 2021-01-24

**Author** Michelle Saul

**Maintainer** Michelle Saul <`msaul@carisls.com`>

**Description** Grows families of features by selecting features that maximize a weighted score calculated from empirical feature scores and graphical knowledge. The final weighted score for a feature is determined by summing a feature's family-weighted scores across all families in which the feature appears.

**License** GPL

**Imports** Rcpp (>= 1.0.6), plyr (>= 1.8.6), stats (>= 3.6.0)

**LinkingTo** Rcpp, RcppArmadillo

**LazyData** true

**NeedsCompilation** yes

**Repository** CRAN

**Date/Publication** 2021-02-05 08:50:08 UTC

## Contents

## `FamilyRank-package` *Algorithm for Ranking Predictors Using Graphical Domain Knowledge*

### Description

Grows families of features by selecting features that maximize a weighted score calculated from empirical feature scores and graphical knowledge. The final weighted score for a feature is determined by summing a feature's family-weighted scores across all families in which the feature appears.

### Details

The DESCRIPTION file:

| | |
|---|---|
| Package: | FamilyRank |
| Type: | Package |
| Title: | Algorithm for Ranking Predictors Using Graphical Domain Knowledge |
| Version: | 1.0 |
| Date: | 2021-01-24 |
| Author: | Michelle Saul |
| Maintainer: | Michelle Saul <msaul@carisls.com> |
| Description: | Grows families of features by selecting features that maximize a weighted score calculated from empirical feat |
| License: | GPL |
| Imports: | Rcpp (>= 1.0.6), plyr (>= 1.8.6), stats (>= 3.6.0) |
| LinkingTo: | Rcpp, RcppArmadillo |
| LazyData: | true |

Index of help topics:

| | |
|---|---|
| `FamilyRank-package` | `Algorithm for Ranking Predictors Using Graphical Domain Knowledge` |
| `createCase` | `Simulate Cases` |
| `createControl` | `Simulate Control` |
| `createData` | `Simulate Data` |
| `createGraph` | `Simulate Graph` |
| `familyRank` | `Feature Ranking with Family Rank` |
| `grow` | `Grow Families` |
| `indexFeats` | `Re-index features` |
| `rbinorm` | `Bimodal Normal Distribution` |

The main function is `familyRank`.

### Author(s)

Michelle Saul

Maintainer: Michelle Saul <msaul@carisls.com>

## References

ADD REFERENCE

---

`createCase` *Simulate Cases*

---

## Description

Numerical feature simulation for positive samples. Called by `createData`.

## Usage

```
createCase(subtype, upper.mean, lower.mean, upper.sd, lower.sd, n.features,
subtype1.feats = 1:5, subtype2.feats = 6:10, subtype3.feats = 11:15)
```

## Arguments

| | |
|---|---|
| `subtype` | Numeric number indicating which subtype to simulate. Currently supports three subtype: 1, 2 or 3. |
| `upper.mean` | The mean of the upper component of the bimodal Gaussian distribution from which features are simulated. |
| `lower.mean` | The mean of the lower component of the bimodal Gaussian distribution from which features are simulated. |
| `upper.sd` | The standard deviation of the upper component of the bimodal Gaussian distribution from which features are simulated. |
| `lower.sd` | The standard deviation of the lower component of the bimodal Gaussian distribution from which features are simulated. |
| `n.features` | Number of features to simulate. |
| `subtype1.feats` | Numeric vector representing the indices of features that define subtype 1. |
| `subtype2.feats` | Numeric vector representing the indices of features that define subtype 2. |
| `subtype3.feats` | Numeric vector representing the indices of features that define subtype 3. |

## Details

Simulations support 3 subtypes, each defined by 5 different features.

Subtype 1 is defined as having the first 3 `subtype1.feats` and at least one of the next 2 `subtype1.feats` simulated from the upper component of the bimodal Gaussian distribution.

Subtype 2 is defined as having all 5 `subtype2.feats` simulated from the upper component.

Subtype 3 is defined as having the first 4 `subtype3.feats` simulated from the upper component and and the last `subtype3.feats` simulated from the lower component.

## Value

Returns a vector of simulated features

### Note

`createCase` is not meant to be called alone. It is designed as a helper function for `createData`.

### Author(s)

Michelle Saul

### References

ADD REFERENCE

### See Also

`createData`

### Examples

```
# Toy Example
case <- createCase(subtype = 1, upper.mean = 13, lower.mean = 5,
upper.sd = 1, lower.sd = 1, n.features = 20,
                        subtype1.feats = 1:5,
                        subtype2.feats = 6:10,
                        subtype3.feats = 11:15)
```

---

## `createControl` *Simulate Control*

### Description

Numerical feature simulation for negative samples. Called by `createData`.

### Usage

```
createControl(upper.mean, lower.mean, upper.sd, lower.sd, n.features,
subtype1.feats = 1:5, subtype2.feats = 6:10, subtype3.feats = 11:15)
```

### Arguments

| | |
|---|---|
| `upper.mean` | The mean of the upper component of the bimodal Gaussian distribution from which features are simulated. |
| `lower.mean` | The mean of the lower component of the bimodal Gaussian distribution from which features are simulated. |
| `upper.sd` | The standard deviation of the upper component of the bimodal Gaussian distribution from which features are simulated. |
| `lower.sd` | The standard deviation of the lower component of the bimodal Gaussian distribution from which features are simulated. |
| `n.features` | Number of features to simulate. |

| | |
|---|---|
| `subtype1.feats` | Numeric vector representing the indices of features that define subtype 1. |
| `subtype2.feats` | Numeric vector representing the indices of features that define subtype 2. |
| `subtype3.feats` | Numeric vector representing the indices of features that define subtype 3. |

## Details

Simulates data such that none of the 3 subtypes defined in `createCase` are represented.

To ensure subtype 1 is not represented, at least one of the first three `subtype1.feats` and/or both of the next 2 `subtype1.feats` are simulated from the lower component of the Gaussian distribution.

To ensure subtype 2 is not represented, at least one of the five `subtype2.feats` is simulated from the lower component.

To ensure subtype 3 is not represented, at least one of the first 4 `subtype3.feats` is simulated from the lower component and/or the last `subtype3.feats` is simulated from the upper component.

## Value

Returns a vector of simulated features

## Note

`createControl` is not meant to be called alone. It is designed as a helper function for `createData`.

## Author(s)

Michelle Saul

## References

ADD REFERENCE

## See Also

`createData`

## Examples

```
# Toy Example
control <- createControl(upper.mean = 13, lower.mean = 5,
upper.sd = 1, lower.sd = 1, n.features = 20,
                        subtype1.feats = 1:5,
                        subtype2.feats = 6:10,
                        subtype3.feats = 11:15)
```

---

## `createData` *Simulate Data*

---

### Description

Simulate data sets meant to emulate gene expression data in oncology.

### Usage

```
createData(n.case, n.control, mean.upper = 13, mean.lower = 5,
sd.upper = 1, sd.lower = 1, n.features = 10000,
subtype1.feats = 1:5, subtype2.feats = 6:10, subtype3.feats = 11:15)
```

### Arguments

| | |
|---|---|
| `n.case` | Number of cases to simulate. |
| `n.control` | Number of controls to simulate |
| `mean.upper` | Mean of upper component of bimodal Gaussian distribution from which features are simulated. |
| `mean.lower` | Mean of lower component of bimodal Gaussian distribution from which features are simulated. |
| `sd.upper` | Standard deviation of upper component of bimodal Gaussian distribution from which features are simulated. |
| `sd.lower` | Standard deviation of lower component of bimodal Gaussian distribution from which features are simulated. |
| `n.features` | Number of features to simulate |
| `subtype1.feats` | Index of features used to define subtype 1. |
| `subtype2.feats` | Index of features used to define subtype 2. |
| `subtype3.feats` | Index of features used to define subtype 3. |

### Details

Simulates case/control data as described in `createCase` and `createControl`, and graphical domain knowledge as described in `createGraph`.

### Value

Returns a named list with a simulated feature matrix (`x`), simulated binary response vector (`y`), vector of subtype labels (`subtype`), and simulated domain knowledge graph (`graph`).

### Author(s)

Michelle Saul

### References

ADD REFERENCE

### See Also

`createCase`, `createControl`, `createGraph`

### Examples

```
## Toy Example
# Simulate data set
# 10 samples
# 20 features
# Features 1 through 15 perfectly define response
# All other features are random noise.
data <- createData(n.case = 5, n.control = 5, mean.upper=13, mean.lower=5,
                   sd.upper=1, sd.lower=1, n.features = 20,
                   subtype1.feats = 1:5, subtype2.feats = 6:10,
                   subtype3.feats = 11:15)
x <- data$x
y <- data$y
graph <- data$graph
```

---

## `createGraph` *Simulate Graph*

### Description

Simulate domain knowledge graph.

### Usage

```
createGraph(subtype1.feats = 1:5, subtype2.feats = 6:10, subtype3.feats = 11:15,
n.interactions = 1e+06, n.features = 10000)
```

### Arguments

| | |
|---|---|
| `subtype1.feats` | Index of features used to define subtype 1. |
| `subtype2.feats` | Index of features used to define subtype 2. |
| `subtype3.feats` | Index of features used to define subtype 3. |
| `n.interactions` | Number of pairwise interactions to simulate. |
| `n.features` | Number of features to simulate |

## Value

Returns a data frame representation of a graph. The first two columns represent graph nodes and the third column represents the edge weights between nodes.

All pairwise combinations of `subtype1.feats` have an edge weight of 1.

All pairwise combinations of `subtype2.feats` have an edge weight of 1.

All pairwise combinations of `subtype3.feats` have an edge weight of 1.

All other pairwise combinations have an edge weight uniformly distributed between 0 and 1.

## Author(s)

Michelle Saul

## References

ADD REFERENCE

## See Also

`createData`

## Examples

```
# Toy Example
graph <- createGraph(subtype1.feats = 1:5, subtype2.feats = 6:10, subtype3.feats = 11:15,
n.interactions = 100, n.features = 20)
```

---

# `familyRank` *Feature Ranking with Family Rank*

## Description

Ranks features by incorporating graphical knowledge to weight empirical feature scores. This is the main function of the FamilyRank package.

## Usage

```
familyRank(scores, graph, d = 0.5, n.rank = min(length(scores), 1000),
n.families = min(n.rank, 1000), tol = 0.001)
```

## Arguments

| | |
|---|---|
| `scores` | A numeric vector of empirical feature scores. Higher scores should indicate a more predictive feature. |
| `graph` | A matrix or data frame representation of a graph object. |
| `d` | Damping factor |
| `n.rank` | Number of features to rank. |
| `n.families` | Number of families to grow. |
| `tol` | Tolerance |

## Details

The `scores` vector should be generated using an existing statistical method. Higher scores should correspond to more predictive features. It is up to the user to adjust accordingly. For example, if the user wishes to use p-values as the empirical score, the user should first adjust the p-values, perhaps by subtracting all p-values from 1, so that a higher value corresponds to a more predictive feature.

The `graph` must be supplied in matrix form, where the first two columns represent graph nodes and the third column represents the edge weights between nodes. The graph nodes must be represented by the index of the feature that corresponds with the index in the `score` vector. For example, a node corresponding to the first value of the `score` vector should be indicated by a 1 in the `graph` object, the second by a 2, etc. It is not necessary that every feature in the `score` vector appear in the `graph`. Missing pairwise interactions will be considered to have interaction scores of 0.

The damping factor, `d`, represents the percentage of weight given to the interaction scores. The damping factor must be between 0 and 1. Higher values give more weight to the interaction score while lower values give more weight to the empirical score.

The value for `n.rank` must be less than or equal to the number of scored features. The algorithm will include only the top `n.rank` features in the ranking process (e.g. the `n.rank` features with the highest values in the `score` vector will be used to grow families). Higher values of `n.rank` require longer compute times.

The value for `n.families` must be less than or equal to the value of `n.rank`. This is the number of families the algorithm will grow. If `n.families` is less than `n.rank`, the algorithm will initate families using the `n.families` highest scoring features. Higher values of `n.families` require longer compute times.

The tolerance variable, `tol`, tells the algorithm when to stop growing a family. Features are added to families until the weighted score is less than the tolerance level, or until all features have been added.

## Value

Returns a vector of the weighted feature scores.

## Author(s)

Michelle Saul

## References

ADD REFERENCE

## Examples

```
# Toy Example
scores <- c(.6, .2, .9)
graph <- cbind(c(1,1), c(2,3), c(.4, .8))
familyRank(scores = scores, graph = graph, d = .5)

# Simulate data set
# 100 samples
# 1000 features
```

```
# Features 1 through 15 perfectly define response
# All other features are random noise
simulatedData <- createData(n.case = 50, n.control = 50, mean.upper=13, mean.lower=5,
                            sd.upper=1, sd.lower=1, n.features = 10000,
                            subtype1.feats = 1:5, subtype2.feats = 6:10,
                            subtype3.feats = 11:15)
x <- simulatedData$x
y <- simulatedData$y
graph <- simulatedData$graph

# Score simulated features using absolute difference in group means
scores <- apply(x, 2, function(col){
  splt <- split(col, y)
  group.means <- unlist(lapply(splt, mean))
  score <- abs(diff(group.means))
  names(score) <- NULL
  return(score)
})

# Display top 15 features using emprical score
order(scores, decreasing = TRUE)[1:15]

# Rank scores using familyRank
scores.fr <- familyRank(scores = scores, graph = graph, d = .5)
# Display top 15 features using emprical scores with Family Rank
order(scores.fr, decreasing = TRUE)[1:15]
```

---

## `grow` *Grow Families*

### Description

Call to the C++ function that grows the families.

### Usage

```
grow(n, f, d, graph, scores, feat_mat, score_mat, tol, weight_mat, selected)
```

### Arguments

| | |
|---|---|
| `n` | Number of features to rank. |
| `f` | Number of families to grow. |
| `d` | Damping factor |
| `graph` | A matrix or data frame representation of a graph object. |
| `scores` | A numeric vector of empirical feature scores. |
| `feat_mat` | Matrix to store selected features. |
| `score_mat` | Matrix to store weighted scores of selected features. |

| | |
|---|---|
| `tol` | Tolerance |
| `weight_mat` | A matrix to store the cumulative weighted scores of selected futures across all families. |
| `selected` | Vector indicating whether a feature has been selected yet. |

## Details

This is the workhorse function for the Family Rank algorithm.

## Value

Returns a matrix with `1+2xn.families` columns and `n.rank` rows. The first column is the cumulative feature score for each of the ranked features `1:n.rank`. The row number corresponds to the re-indexed feature index. The next `n.families` columns contain the indices of selected features for each iteration of feature selection. The last `n.families` columns contain the weigthed scores of selected features for each iteration.

## Author(s)

Michelle Saul

## References

ADD REFERENCE

## Examples

```
# Toy Example
scores <- c(.6, .2, .9)
graph <- cbind(c(1,1), c(2,3), c(.4, .8))

# initialize matrices
n <- n.families <- length(scores)
feat.mat <- score.mat <- matrix(0, nrow = n, ncol = n.families)
feat.mat[1,] <- order(scores, decreasing = TRUE)
score.mat[1,] <- sort(scores, decreasing = TRUE)

# Grow families
mats <- grow(n = n, f = n.families, d = 0.5, graph = as.matrix(graph),
             scores = scores,
             feat_mat = feat.mat, score_mat = score.mat, tol = 0,
             weight_mat = as.matrix(scores), selected = rep(1, n))
# Selected Feature Matrix
## columns represent familes
## rows represent iterations
## values indicate indices of selected features
feat.mat <- mats[, 2:(n.families+1)]
feat.mat
# Corresponding Score Matrix
## columns represent familes
## rows represent iterations
```

```
## values indicate max weighted score of selected features
score.mat <- mats[, (n.families+2):(1+2*n.families)]
score.mat
```

---

## `indexFeats` *Re-index features*

---

### Description

Re-index features based on number to rank. Called by `familyRank`.

### Usage

```
indexFeats(scores, graph, n.rank = NULL)
```

### Arguments

| | |
|---|---|
| `scores` | A numeric vector of empirical feature scores. |
| `graph` | A matrix or data frame representation of a graph object. |
| `n.rank` | Number of features to rank. |

### Details

This function is used to re-index features for the Family Rank algorithm. The function takes in the scores for all features, and returns scores for the top `n.rank` features. It also takes in the full domain knowledge `graph` and returns the subgraph that only includes interactions between the top `n.rank` features. Finally, it re-indexes the top features in both the score vector and domain knowledge graph to 1:n.rank.

### Value

Returns a named list with re-indexed domain knowledge graph (`graph.w`), re-indexed scores (`score.w`), a mapping between original and new indices (`loc.map`), and the number of features to rank (`n.rnak`).

### Note

`indexFeats` is not meant to be called alone. It is designed as a helper function for `familyRank`.

### Author(s)

Michelle Saul

### References

ADD REFERENCE

### See Also

`familyRank`

---

`rbinorm` *Bimodal Normal Distribution*

---

## Description

Simulates random data from a bimodal Gaussian distribution.

## Usage

```
rbinorm(n, mean1, mean2, sd1, sd2, prop)
```

## Arguments

| | |
|---|---|
| `n` | Number of observations to simulate |
| `mean1` | Mean of mode 1 |
| `mean2` | Mean of mode 2 |
| `sd1` | Standard deviation of mode 1 |
| `sd2` | Standard deviation of mode 2 |
| `prop` | Probability of being in mode 1. `1 - prop` is the probability of being in mode 2. |

## Details

This function is modeled off of the `rnorm` function.

## Value

Generates random deviates

## Author(s)

Michelle Saul

## Examples

```
## Generate 100 samples from a two component Guassian curve
samples <- rbinorm(n=100, mean1=10, mean2=20, sd1=1, sd2=2, prop=.5)

## Plot distribution of simulated data
plot(density(samples))
```

# Index