Package 'miRNAss'

May 8, 2026

Type Package

Title Genome-Wide Discovery of Pre-miRNAs with few Labeled Examples

Version 1.5

Author Cristian Yones

Maintainer Cristian Yones <cyones@sinc.unl.edu.ar>

Description Machine learning method specifically designed for pre-miRNA prediction. It takes advantage of unlabeled sequences to improve the prediction rates even when there are just a few positive examples, when the negative examples are unreliable or are not good representatives of its class. Furthermore, the method can automatically search for negative examples if the user is unable to provide them. MiRNAss can find a good boundary to divide the pre-miRNAs from other groups of sequences; it automatically optimizes the threshold that defines the classes boundaries, and thus, it is robust to high class imbalance. Each step of the method is scalable and can handle large volumes of data.

License Apache License 2.0

Encoding UTF-8

LazyData true

Imports Matrix, stats, Rcpp, CORElearn, RSpectra,

LinkingTo Rcpp

NeedsCompilation yes

RoxygenNote 6.0.1

Repository CRAN

Date/Publication 2020-10-20 18:20:02 UTC

Contents

adjacencyMatrixKNN *MiRNAss: Genome-wide pre-miRNA discovery from few labeled examples*

Description

This funtions builds the adjacency matrix (the graph) given a data frame of numerical features.

Usage

```
adjacencyMatrixKNN(sequenceFeatures, sequenceLabels = rep(0,
  nrow(sequenceFeatures)), nNearestNeighbor = 10, threadNumber = NA)
```

Arguments

sequenceFeatures	Data frame with features extracted from stem-loop sequences.
sequenceLabels	Vector of labels of the stem-loop sequences. It must have -1 for negative examples, 1 for known miRNAs and zero for the unknown sequences (the ones that would be classificated).
nNearestNeighbor	Number of nearest neighbors in the KNN graph. The default value is 10.
threadNumber	Number of threads used for the calculations. If it is NA leave OpenMP decide the number (may vary across different platforms).

Value

Returns the eigen descomposition as a list with two elements: The eigen vectors matrix 'U' and the eigen values vector 'D'.

Examples

```
# First construct the label vector with the CLASS column
y = as.numeric(celegans$CLASS)*2 - 1

# Remove some labels to make a test
y[sample(which(y>0),200)] = 0
y[sample(which(y<0),700)] = 0

# Take all the features but remove the label column
x = subset(celegans, select = -CLASS)

A = adjacencyMatrixKNN(x, y, 10, 8)

for (nev in seq(50,200, 50)) {
    # the data frame of features 'x' should not be pass as parameter
    p = miRNAss(sequenceLabels = y, AdjMatrix = A,
                nEigenVectors = nev)
```

```
    # Calculate some performance measures
    SE = mean(p[ celegans$CLASS & y==0] > 0)
    SP = mean(p[!celegans$CLASS & y==0] < 0)
    cat("N: ", nev, "\n  SE: ", SE, "\n  SP:   ", SP, "\n")
}
```

`celegans` *Features extracted from hairpins of Caenorhabditis elegans.*

Description

Small dataset of features extracted from C. elegans hairpins. The full dataset is contained in the zip file "experiment_scripts.zip" that can be downloaded from:

Usage

```
celegans
```

Format

A data frame with 1000 rows and 29 columns. The first 28 columns are numeric features used in [1]. The last column is a logical variable indicating if the stem-loop is a pre-miRNA or not.

Details

http://sourceforge.net/projects/sourcesinc/files/mirnass/

References

[1] Gudyś, A., Szcześniak, M. W., Sikora, M., & Makałowska, I. (2013). HuntMi: an efficient and taxon-specific approach in pre-miRNA identification. BMC bioinformatics, 14(1), 1.

`eigenDecomposition` *MiRNAss: Genome-wide pre-miRNA discovery from few labeled examples*

Description

This funtions calculate the eigenvectors and eigen values of the Laplacian of the graph. As this proccess is quite time comsumin, this functions allows to obtain this decomposition once and the be able to run miRNAss several times in shorter times.

Usage

```
eigenDecomposition(AdjMatrix, nEigenVectors)
```

Arguments

AdjMatrix	Adjacency sparse matrix of the graph.
nEigenVectors	Number of eigen vectors.

Value

Returns the eigen descomposition as a list with two elements: The eigen vectors matrix 'U' and the eigen values vector 'D'.

Examples

```
# First construct the label vector with the CLASS column
y = as.numeric(celegans$CLASS)*2 - 1

# Remove some labels to make a test
y[sample(which(y>0),200)] = 0
y[sample(which(y<0),700)] = 0

# Take all the features but remove the label column
x = subset(celegans, select = -CLASS)

A = adjacencyMatrixKNN(x, y, 10, 8)
E = eigenDecomposition(AdjMatrix = A, nEigenVectors = 100)
for (mp in c(0.1,1,10)) {
    p = miRNAss(sequenceLabels = y, AdjMatrix = A,
                eigenVectors = E, missPenalization = mp)
    # Calculate some performance measures
    SE = mean(p[ celegans$CLASS & y==0] > 0)
    SP = mean(p[!celegans$CLASS & y==0] < 0)
    cat("mP: ", mp, "\n  SE: ", SE, "\n  SP:   ", SP, "\n")
}
```

`miRNAss` *MiRNAss: Genome-wide pre-miRNA discovery from few labeled examples*

Description

This is the main function of the miRNAss package and implements the miRNA prediction method, It takes as main parameters a matrix with numerical features extracted from RNA hairpins and an incomplent vector of labels where the positive number represents known miRNAs, the negative are not-miRNA hairpins and te zero values are unknown sequences (those that will be classified). As a results it returns a complete label vector.

Usage

```
miRNAss(sequenceFeatures = NULL, sequenceLabels, AdjMatrix = NULL,
  nNearestNeighbor = 10, missPenalization = 1, scallingMethod = "relief",
  thresholdObjective = "Gm", neg2label = 0.05, positiveProp = NULL,
  eigenVectors = NULL, nEigenVectors = min(400,
  round(length(sequenceLabels)/5)), threadNumber = NA)
```

Arguments

`sequenceFeatures`
: Data frame with features extracted from stem-loop sequences. It is not required if the adjacency matrix is provided.

`sequenceLabels`
: Vector of labels of the stem-loop sequences. It must have -1 for negative examples, 1 for known miRNAs and zero for the unknown sequences (the ones that would be classificated).

`AdjMatrix`
: Sparse adjacency matrix representeing the graph. If sequence features are provided it is ignored.

`nNearestNeighbor`
: Number of nearest neighbors in the KNN graph. The default value is 10.

`missPenalization`
: Penalization of the missclassification of known examples. The default value is 1. If the examples are not very confident, this value can be diminished.

`scallingMethod`
: Method used for normalization and scalling of the features. The options are 'none', 'whitening' and 'relief' (the default option). The first option does nothing, the second calls the built-in function 'scale' and the last one uses the ReliefFexpRank algorithm from the coreLearn package.

`thresholdObjective`
: Performance measure that would be optimized when estimating the threshold. The options are 'Gm' (geometric mean of the SE and the SP), 'G' (geometric mean of the SE and the precision), 'F1' (harmonic mean between SE and the precision) and 'none' (do not calculate any threshold). The default value is 'Gm'.

`neg2label`
: Proportion of unlabeled stem-loops that would be labeled as negative with the automatic method to start the classification algorithm. The default is 0.05.

`positiveProp`
: Expected proportion of positive sequences. If it is not provided by the user, is estimated as sum(y > 0) / sum(y != 0) when there are negative examples or as 2 * sum(y > 0) / sum(y == 0) when not.

`eigenVectors`
: Eigen decomposition of the Laplacian matrix, as returned by the function eigen-Decomposition. If is not provided is calculated internally (this parameter allows to calculate the eigen vectors once and then run several times miRNAss with the same eigen vectors).

`nEigenVectors`
: Number of eigen vectors used to aproximate the solution of the optimization problem. If the number is too low, smoother topographic solutions are founded, probabily losing SP but achieving a better SE. Generally, 400 are enought.

`threadNumber`
: Number of threads used for the calculations. If it is NA leave OpenMP decide the number (may vary across different platforms).

Value

Returns a vector with the same size of the input vector y with the prediction scores for all sequences (even the labelled examples). If a threshold Objective different from 'none' was set, the threshold is estimated and subtracted from the scores, therefore the new threshold that divide the classes is zero. Also, the positive scores are divided by the max positive score, and the negative scores are divided by the magnitud of the minimum negative score.

Examples

```
# First construct the label vector with the CLASS column
y = as.numeric(celegans$CLASS)*2 - 1

# Remove some labels to make a test
y[sample(which(y>0),200)] = 0
y[sample(which(y<0),700)] = 0

# Take all the features but remove the label column
x = subset(celegans, select = -CLASS)

# Call miRNAss with default parameters
p = miRNAss(x,y)

# Calculate some performance measures
SE = mean(p[ celegans$CLASS & y==0] > 0)
SP = mean(p[!celegans$CLASS & y==0] < 0)
cat("Sensitivity: ", SE, "\nSpecificity: ", SP, "\n")
```

Index