

Package ‘mixfdr’

February 14, 2012

Type Package

Title Computes false discovery rates and effect sizes using normal mixtures

Version 1.0

Date 2009-07-21

Author Omkar Muralidharan, with many suggestions from Bradley Efron

Maintainer Omkar Muralidharan <omkar@stanford.edu>

Description This package fits normal mixture models to data and uses them to compute effect size estimates and local and tail area false discovery rates. To make this precise, suppose you have many normally distributed z 's, and each $z[i]$ has mean $\delta[i]$. This package will estimate $\delta[i]$ based on the z 's (effect sizes), $P(\delta[i]=0|z[i])$ (local false discovery rates) and $P(\delta[i]=0||z[i]|>z)$ (tail area false discovery rates).

License GPL-2

LazyLoad yes

Repository CRAN

Date/Publication 2009-07-21 15:17:46

R topics documented:

mixfdr-package	2
calibrateP	2
dens.mixture	4
mixFdr	6

Index	10
--------------	-----------

mixfdr-package *Computes false discovery rates and effect sizes using normal mixtures*

Description

This package fits normal mixture models to data and uses them to compute effect size estimates and local and tail area false discovery rates. To make this precise, suppose you have many normally distributed z 's, and each $z[i]$ has mean $\delta[i]$. This package will estimate $\delta[i]$ based on the z 's (effect sizes), $P(\delta[i]=0|z[i])$ (local false discovery rates) and $P(\delta[i]=0||z[i]|>z)$ (tail area false discovery rates).

Details

Package: mixfdr
Type: Package
Version: 1.0
Date: 2009-07-21
License: GPL-2
LazyLoad: yes

See the help file for [mixFdr](#) for an explanation and common usage.

Author(s)

Omkar Muralidharan, with many suggestions from Bradley Efron
Maintainer: Omkar Muralidharan
<omkar@stanford.edu>

References

See papers at the author's website <http://stat.stanford.edu/~omkar>

calibrateP *Internal Utility Functions*

Description

These functions are used by mixfdr to calculate various things. calibrateP is used to calibrate the penalization, and getStartsNew gets a reasonable starting point for the fitting. matDens computes a density matrix. mixModelFitter is the workhorse model fitter, and mixModelManyStarts is used to try various starting points.

Usage

```

calibrateP(m, B = 25)
getStartsNew(x, J, noiseSD = 1)
matDens(x, mu, sig)
mixModelFitter(x, J, N, starts, maxiter, tol, p, muInt, sigInt)
mixModelManyStarts(x, J, N, starts = NA, maxiter = 1000, tol = 0.001, p = NA, muInt = NA, sigInt = 1, theonull)

```

Arguments

m	A mixture model, that is, a list with pi, mu and sigma vectors of the same length, and a data vector
B	Number of bootstrap datasets to generate for each candidate penalization
x	Data to fit on. For matDens, any vector of points to calculate the densities on
mu	A vector of means
sig	A vector of standard deviations. Should be the same length as mu
J	Number of mixture components
noiseSD	The sampling standard deviation
N	Somewhat confusingly, the penalization parameter. Called P by functions intended for user use
starts	A matrix of starts. The matrix should be J x 3, with column 1 for pi, 2 for mu, and 3 for sigma. For mixModelManyStarts, this should be an array, with dimension (num starts, J, 3).
maxiter	Max number of iterations
tol	Convergence tolerance
p	Penalization vector. Must be of length J. Nearly always (1,0,...,0)
muInt	mu constraints, a J x 2 matrix. mu[j] will be forced to lie in [mu[j, 1], mu[j, 2]]. Inf and -Inf allowed. If missing, each row will be [-Inf, Inf]
sigInt	sigma constraints, similar to muInt. First column should be positive. For mixModelManyStarts, if a constant s, each row will be [s, Inf]
theonull	Use the theoretical null? If true, sets muInt[1,] = 0 and sigInt[1,] = 1

Details

These functions are not intended for user use, with the possible exception of calibrateP. Most people will want to use [mixFdr](#) and/or one of the utilities like [effectSize](#).

calibrateP perturbs the null standard deviation of the supplied model and fits many models with many penalizations to try and see which penalization is best when the true model is "close" to the supplied model. It can be quite slow.

getStartsNew finds a decent starting point by dividing up the data into J quantile groups, with the middle group larger than the others. It then calculates the within group mean and sd to use as mu and sigma starting points.

matDens is a faster way to compute $d_{norm}(x[i], \mu[j], \text{sig}[j])$ for all i and j

mixModelFitter fits the mixture model with the EM algorithm. It is fairly quick.

mixModelManyStarts fits many mixture models (with different starting points) and chooses the one with the highest penalized likelihood.

Value

calibrateP returns a penalization. getStartsNew returns a start matrix. matDens returns a matrix M with $M[i, j] = \text{dnorm}(x[i], \mu[j], \text{sig}[j])$. mixModelFitter and mixModelManyStarts return mixture models - lists with π , μ , σ , and data elements.

Author(s)

Omkar Muralidharan

References

See papers at the author's website <http://stat.stanford.edu/~omkar>

See Also

[mixfdr](#), [effectSize](#)

dens.mixture

Calculate fdrs/effect sizes/densities from a mixture model

Description

These functions are used to calculate various quantities derived from a mixture model (fdr, FDR, effectSize, etc.). They are useful if you have a mixture model and you want to calculate, for instance, the fdr estimate on a new data point. They are also very useful for plotting.

Usage

```
dens.mixture(x, m)
groupProbs(z, m)

effectSize(z, m, noiseSD = NA)
fdrMixModel(z, m, nullGroups = NA)
tailFDRMixModel(z, m, nullGroups = NA)

plot.mix.dens(x, plot.subdens = TRUE, ...)
```

Arguments

x	A vector of new data points for which things are to be calculated.
z	A vector of new data points for which things are to be calculated. The same as x, just different notation.
m	A mixture model, as returned by mixFdr . You can make your own: it should be a list with elements π , μ , σ . Each should be a J vector with the appropriate parameter values. Some functions also need another element data, which contains the data the model was fitted on.

<code>nullGroups</code>	A J-length boolean vector indicating which groups should be counted in the null. If NA just takes the first group
<code>noiseSD</code>	The sampling variance (see details for where this comes in the model). If NA, taken from <code>m\$noiseSD</code> if present there, otherwise <code>min(m\$sig)</code> .
<code>plot.subdens</code>	Plot the subdensities?
<code>...</code>	Arguments to pass to <code>hist</code> . Note <code>prob=TRUE</code> is already being passed.

Details

`dens.mixture` computes the mixture density at each `x[i]`

`groupProbs` computes the posterior probability that each `z[i]` came from each group of the mixture model.

`effectSize` computes the posterior mean and variance when `z[i]` is observed. The model is that `z[i]` is $\text{Normal}(\text{delta}[i], \text{noiseSD})$, and `delta[i]` have a normal mixture prior (so that `z[i]` have the marginal density given by `m`). `effectSize` calculates the posterior mean and variance for `delta[i]` given `z[i]`. If `noiseSD` is too big (bigger than any of `m$sig`) then `effectSize` basically increases those `m$sig` to be at least `noiseSD` and returns a warning.

`fdrMixModel` calculates the local false discovery rate (fdr). This is $P(\text{delta}[i]=0|z[i])$, or more generally $P(\text{delta}[i] \text{ null}|z[i])$.

`tailFDRMixModel` calculates the tail area false discovery rate (FDR). This is $P(\text{delta}[i]=0|z[i]>z)$ (two-sided) or $P(\text{delta}[i]=0|z[i]>z)$ (right) or $P(\text{delta}[i]=0|z[i]<z)$ (left). `tailFDRMixModel` computes all three.

`plot.mix.dens` plots a histogram of `m$data` and overlays the density curve (with the group subdensities if required).

Value

`dens.mixture` returns a vector of density values.

`groupProbs` returns a `length(z)` by J matrix of posterior probabilities.

`effectSize` returns a matrix, first column effect sizes, second column posterior variances.

`fdrMixModel` returns a vector of fdr's.

`tailFDRMixModel` returns a list. `FDRTwoSided`, `FDRleft`, `FDRright` have two-sided, left and right FDRs respectively.

Author(s)

Omkar Muralidharan

References

See papers at the author's website <http://stat.stanford.edu/~omkar>

See Also

[mixFdr](#)

Examples

```
z = rnorm(10000)
m = mixFdr(z, plots = FALSE)
s = seq(-5,5,by=0.01)
plot(s, dens.mixture(s,m), t = 'l', main = "Mixture Density")
plot(s, groupProbs(s,m)[,1], t = 'l', main = "Prob of being from group 1")
plot(s, effectSize(s,m)[,1], t = 'l', main = "Effect Size")
plot(s, effectSize(s,m)[,2], t = 'l', main = "Posterior Effect Var")
plot(s, fdrMixModel(s,m), t = 'l', main = "fdr")
plot(s, tailFDRMixModel(s,m)$FDRT, t = 'l', main = "two-sided FDR")
plot.mix.dens(m, TRUE, br = 100)
```

 mixFdr

Fit a mixture model and use it to estimate fdr/FDR/effect sizes

Description

This is the main function for this package. It fits a normal mixture model and uses it to estimate fdr's, FDR's and effect sizes.

Usage

```
mixFdr(x, J = 3, P = NA, noiseSD = NA, theonull = FALSE, calibrate = FALSE, plots = TRUE, nearlyNull = 0.2)
```

Arguments

x	The data. A numeric vector. For example, a vector of transformed t-statistics.
J	The number of mixture components.
P	The penalization parameter. If NA and calibrate is FALSE, will be length(x)/5. If calibrate = TRUE, chosen by calibration. A higher P pushes the model toward one big null group, so the center is fit more closely.
noiseSD	The sampling noise (see details). For transformed t-statistics, or transformed p-values, take this to be 1 at least to start. If NA, fit using a combination of the mixture model and the median of absolute deviations estimator. Note that it is hard to estimate both noiseSD and the empirical null accurately at the same time. If your data is overdispersed, and you try to estimate both, you'll probably end up with noiseSD being about as large as the empirical null sd. This is not a problem for fdr/FDR's but it can be a problem for effect sizes (if the signal is known to be sparse, it's ok).
theonull	Use the theoretical N(0,1) null?
calibrate	Choose P by calibration? Note that calibration can be quite slow.
plots	Produce plots?
nearlyNull	How big does abs(mu[j]) need to be for a group to be considered non-null? 0 will ensure that only the first group is null.

starts	An optional matrix of starting points. This is passed straight to mixModelManyStarts , see that page for more information.
p	The penalization shape parameter, a vector of length J. The penalty on the mixture proportions is Dirichlet(P*p). Nearly always (1,0,...,0)
maxIter	Maximum number of iterations.
tol	Convergence tolerance
nocheck	If FALSE, mixFdr will fit an empirical null model and warn you if the fitted model is very different. Set TRUE if you know what you're doing and need things to be faster.

Details

This function estimates effect sizes, fdr's and FDR's. If you don't want to get into the details, run it with defaults (perhaps using `theonull` to fit a theoretical or empirical null). The object returned has the estimates, see "Value".

Here is a short explanation of the mixture model - for a full account, see the papers on the author's website. Suppose we have $z[i] \sim \text{Normal}(\delta[i], \text{noiseSD})$. We want to estimate $\delta[i]$, $P(\delta[i]=0|z[i])$, and $P(\delta[i]=0||z[i]|>z)$ (effect size, fdr, FDR). Given a prior for δ we can do this. The mixture model *fits* the prior based on the data, using a J group normal mixture model. A theoretical null means the first mixture component is a point mass at 0. An empirical null allows the first mixture component to vary. The mixture proportions are given a Dirichlet(P*p) penalty to stabilize them and to fit empirical nulls if required.

The upshot here is that no matter what the options, the density corresponding to the fitted model needs to fit the data well for any inference to be trustworthy. Look at the histogram plot produced by mixFdr. If the density doesn't fit well, particularly near the center, try using an empirical null (`theonull=FALSE`), increasing the number of groups, calibrating the penalization (or experimenting with it).

If you need to incorporate mixFdr's estimates into another procedure, mixFdr may be too slow. See [mixModelFitter](#) for the workhorse fitting function, which is much faster.

Value

A list of objects. They can be divided into the fitted model, and derived estimates. *The fitted model parameters are for the marginals density NOT the prior on delta.*

pi	Mixture proportions
mu	Mean of each mixture component
sigma	Sd of each mixture component
noiseSD	Fitted or supplied sampling SD
converged	Did the EM converge?
nIter	How many iterations
fdr	local false discovery rate estimates for each case
FDRTwoSided	Two sided FDR estimates for each case
FDRLeft	Left-tail FDR estimates for each case

FDRRight	Right-tail FDR estimates for each case
effectSize	Effect size estimates for each case
effectPostVar	Estimates of the posterior variance of the effect size for each case
call	The call
data	The supplied data

Author(s)

Omkar Muralidharan

References

See papers at the author's website <http://stat.stanford.edu/~omkar>

See Also

[effectSize](#), [mixModelFitter](#)

Examples

```
## A simple workflow

z = c(rnorm(10000, 0.1, 1.2), rnorm(1000, 3, 1)) # null, alternative
m = mixFdr(z, theonull = TRUE)

# uh-oh, warning
# Just visually, the fit is not so good near the center
# Let's try an empirical null

m = mixFdr(z)

# Better. But see how the effect size estimate changes with noiseSD.

# We usually want:
m$fdr
m$effectSize
m$FDRTwoSided

# Suppose we have data with known noise level 1

z = c(rnorm(1000), rnorm(100,3,1))
m = mixFdr(z, J = 3, noiseSD = 1)
delta = m$effectSize
fdrs = m$fdr

# Suppose we wanted fdr and effect size curves
# then we could do

s = seq(-5,5,by=0.01)
effCurve = effectSize(s,m)[,1]
```

```
fdrCurve = fdrMixModel(s,m,abs(m$mu - m$mu[1])<=0.2) # this tells it which indices to consider null

# compare to the Bayes estimator for this problem
mTrue = list(pi = c(10,1)/11, mu = c(0,3), sig = c(1,1))
# note that the model parameters are in terms of the MARGINAL not of the prior
trueEff = effectSize(s,mTrue)[,1]
trueFdr = fdrMixModel(s,mTrue,c(TRUE,FALSE))
par(mfrow = c(2,1))
plot(s, trueEff, t = 'l', main = "Effect Size - Black is true", xlab = "z", ylab = "E(delta|z)", ylim = c(-3,3))
lines(s, effCurve, col = 2)
plot(s, trueFdr, t = 'l', main = "fdr - Black is true", xlab = "z", ylab = "fdr", ylim = c(0,1))
lines(s, fdrCurve, col = 2)
```

Index

*Topic **models**

- calibrateP, 2
- dens.mixture, 4
- mixFdr, 6
- mixfdr-package, 2

*Topic **package**

- mixfdr-package, 2

*Topic **smooth**

- dens.mixture, 4
- mixFdr, 6
- mixfdr-package, 2

calibrateP, 2

dens.mixture, 4

effectSize, 3, 4, 8

effectSize (dens.mixture), 4

fdrMixModel (dens.mixture), 4

getStartsNew (calibrateP), 2

groupProbs (dens.mixture), 4

matDens (calibrateP), 2

mixFdr, 2–5, 6

mixfdr, 4

mixfdr (mixfdr-package), 2

mixfdr-package, 2

mixModelFitter, 7, 8

mixModelFitter (calibrateP), 2

mixModelManyStarts, 7

mixModelManyStarts (calibrateP), 2

plot.mix.dens (dens.mixture), 4

tailFDRMixModel (dens.mixture), 4