epigenomix — Epigenetic and gene transcription data normalization and integration with mixture models

Hans-Ulrich Klein, Martin Schäfer

December 11, 2015

Contents

1 Introduction

This package provides methods for an integrative analysis of gene transcription and epigenetic data, especially histone ChIP-seq data [**?**]. Histone modifications are an epigenetic key mechanism to activate or repress the transcription of genes. Several data sets consisting of matched transcription data and histone modification data localized by ChIP-seq have been published. However, both data types are often analysed separately and results are compared afterwards. The methods implemented here are designed to detect transcripts that are differentially transcribed between two conditions due to an altered histone modification and are suitable for very small sample sizes. Transcription data may be obtained by microarrays or RNA-seq.
Briefly, the following workflow is described in this document:

1. Matching of both data types by assigning the number of ChIP-seq reads aligning within the promoter region to the respective transcription value

2. Normalization of ChIP-seq values

3. Calculation of a correlation score for each gene by multiplying the standardized difference of ChIP-seq values by the standardized difference of transcription values

4. Fitting a (Bayesian) mixture model to this score: The implicit assignment of transcripts to mixture components is used to classify transcripts into one of the following groups: (i) Transcripts with equally directed differences in both data sets, (ii) transcripts with reversely directed differences in both data sets and (iii) transcripts with no differences in at least one of the two data sets. Group (iii) is represented by centred normal components whereas an exponential component is used for group (i) and a mirrored exponential component for group (ii).

2 Data preprocessing and normalization

2.1 Microarray gene expression data

First, we load an example microarray gene expression data set. The data set consists of four samples. Two wild type replicates and two *CEBPA* knock-out replicates. The differences between *CEBPA* knock-down and wild type samples are of interest. The data set is stored as an *ExpressionSet* object and was reduced to a few probesets on chromosome 1.

```
> library(epigenomix)
> data(eSet)
> pData(eSet)
```

```
           CEBPA
CEBPA_WT_a    wt
CEBPA_WT_b    wt
CEBPA_KO_a    ko
CEBPA_KO_b    ko
```

Data was measured using Affymetrix Mouse Gene 1.0 ST arrays and RMA normalization was applied. See packages *affy* and *Biobase* how to process affymetrix gene expression data.

2.2 RNA-seq data

Using RNA-seq instead of microarrays has the advantage that the abundance of individual transcript can be estimated. For this task, software like Cufflinks [**?**] can be employed. Moreover, the Cuffdiff method (part of the Cufflinks software package) allows to summarize the estimated transcript abundances over all transcripts that share the same transcriptional start site (TSS) and offers several normalization methods, e.g. scaling based on the observed quartiles [**?**]. Grouping all transcripts sharing the same TSS is favourable for the later matching task. Importing the Cuffdiff output as data frame gives us the FPKM (fragments per kilobase of transcript per million fragments mapped) values.

```
> data(fpkm)
> head(fpkm[c(-2,-8), ])
```

```
    tracking_id     gene_id gene_short_name
4       TSS1000 XLOC_000367        SH3BGRL3
38     TSS10003 XLOC_003811           TMCO1
49     TSS10004 XLOC_003812   RP11-525G13.2
82     TSS10007 XLOC_003814          FAM78B
149    TSS10013 XLOC_003815     RP11-9L18.2
160    TSS10014 XLOC_003816    RP11-479J7.1
                     locus   CEBPA_WT    CEBPA_KO   tss_id
4     1:26605666-26647014 9.01200000 6.54111e+01  TSS1000
```

```
38  1:165696031-165880855 0.05631180 5.08823e-02 TSS10003
49  1:165696031-165880855 1.08156000 8.27812e-01 TSS10004
82  1:166026673-166136206 0.00160972 2.34348e-03 TSS10007
149 1:166244865-166246834 1.15553000 1.48034e+00 TSS10013
160 1:166356963-166421869 0.00128422 8.06234e-04 TSS10014
                         ensemble.estid chr     start
4                     ENST00000319041.6   1  26606613
38  ENSG00000143183.12,ENST00000580248.1   1 165696032
49   ENST00000455257.1,ENSG00000236364.1   1 165865116
82   ENSG00000188859.5,ENST00000338353.3   1 166026674
149  ENST00000400979.2,ENSG00000215835.2   1 166244866
160  ENST00000448643.1,ENSG00000225325.1   1 166356964
          end strand       tss
4    26607941      +  26606613
38  165796992      - 165796992
49  165869592      - 165869592
82  166136206      - 166136206
149 166246834      - 166246834
160 166421869      - 166421869
```

The last six columns were not included in the Cuffdiff output, but were extracted from the annotation file given as input to Cuffdiff. Next, we construct an *ExpressionSet* object so that we can handle RNA-seq data in the same way as microarray data:

```
> mat <- log2(as.matrix(fpkm[, c("CEBPA_WT", "CEBPA_KO")]))
> rownames(mat) <- fpkm$tss_id
> eSet.seq <- ExpressionSet(mat)
> pData(eSet.seq)$CEBPA <- factor(c("wt", "ko"))
> fData(eSet.seq)$chr <- fpkm$chr
> fData(eSet.seq)$tss <- fpkm$tss
```

2.3 Histone ChIP-seq data

The example histone ChIP-seq data is stored as *GRangesList* object:

```
> data(mappedReads)
> names(mappedReads)
```

```
[1] "CEBPA_WT_1" "CEBPA_KO_1"
```

There are two elements within the list. One *CEBPA* wild type and one knock-out sample. Most of the originally obtained reads were removed to reduce storage space. Further, the reads were extended towards the 3 prime end to the mean DNA fragment size of 200bps and duplicated reads were removed. See R packages *Rsamtools* and *GenomicAlignments* how to read in and process sequence reads

2.4 Data matching

The presented ChIP-seq data lozalized H3K4me3 histone modifications. This modification primarily occures at promoter regions. Hence, we assign ChIP-seq values to transcription values by counting the number of reads lying wihtin the promoter of the measured transcript.

2.4.1 Microarray gene expression data

Depending on the array design, probes often measure more than one transcripts simultaneously. These transcripts may have different TSS/promoters. This makes data matching in case of arrays somewhat tricky. We first create a list with one element for each probe that stores the Ensemble transcript IDs of all transcripts measured by that probeset:

```
> probeToTrans <- fData(eSet)$transcript
> probeToTrans <- strsplit(probeToTrans, ",")
> names(probeToTrans) <- featureNames(eSet)
```

Next, we need the transcriptional start sites for each transcript.

```
> data(transToTSS)
> head(transToTSS)
```

```
     ensembl_transcript_id chromosome_name transcript_start
159     ENSMUST00000001172               1         36547201
441     ENSMUST00000003219               1         39535802
631     ENSMUST00000004829               1        171559193
766     ENSMUST00000006037               1         13374083
1202    ENSMUST00000013842               1        172206804
1306    ENSMUST00000015460               1        171767127
     strand
159      -1
441       1
631       1
766      -1
1202     -1
1306      1
```

Such a data frame can be obtained e.g. using *biomaRt*:

```
> library("biomaRt")
> transcripts <- unique(unlist(probeToTrans))
> mart <- useMart("ENSEMBL_MART_ENSEMBL",dataset="mmusculus_gene_ensembl", host="www.ensembl
> transToTSS <- getBM(attributes=c("ensembl_transcript_id",
     "chromosome_name", "transcript_start",
     "transcript_end", "strand"),
     filters="ensembl_transcript_id",
```

```
    values=transcripts, mart=mart)
> indNeg <- transToTSS$strand == -1
> transToTSS$transcript_start[indNeg] <- transToTSS$transcript_end[indNeg]
> transToTSS$transcript_end <- NULL
```

Having these information, the promoter region for each probe can be calculated unsing `matchProbeToPromoter`. Argument `mode` defines how probes with multiple transcripts should be handled.

```
> promoters <- matchProbeToPromoter(probeToTrans,
    transToTSS, promWidth=6000, mode="union")
> promoters[["10345616"]]
```

```
GRanges object with 2 ranges and 1 metadata column:
      seqnames               ranges strand |       probe
         <Rle>            <IRanges>  <Rle> | <character>
  [1]        1 [37869206, 37875205]      + |    10345616
  [2]        1 [37887407, 37893406]      - |    10345616
  -------
  seqinfo: 1 sequence from an unspecified genome; no seqlengths
```

Note that some promoter regions, like for probeset `"10345616"`, may consist of more than one interval.
Finally, `summarizeReads` is used to count the number of reads within the promoter regions:

```
> chipSetRaw <- summarizeReads(mappedReads, promoters, summarize="add")
> chipSetRaw
```

```
class: ChIPseqSet
dim: 180 2
metadata(0):
assays(1): chipVals
rownames(180): 10344803 10344813 ... 10361191
  10361215
rowData names(0):
colnames(2): CEBPA_WT_1 CEBPA_KO_1
colData names(1): totalCount
```

```
> head(chipVals(chipSetRaw))
```

```
         CEBPA_WT_1 CEBPA_KO_1
10344803        145        401
10344813        145        401
10344897          2          8
10345007          8          6
10345037         69        122
10345099         38         90
```

The method returns an object of class *ChIPseqSet*, which is derived from class *RangedSummarizedExperiment*.

2.4.2 RNA-seq data

In case of RNA-seq data, we have one transcription value for each group of transcripts sharing the same TSS. Hence, a promoter region can be simply assigned to each transcription value:

```
> promoters.seq <- GRanges(seqnames=fData(eSet.seq)$chr,
                           ranges=IRanges(start=fData(eSet.seq)$tss, width=1),
                           probe=featureNames(eSet.seq))
> promoters.seq <- resize(promoters.seq, width=3000, fix="center")
> promoters.seq <- split(promoters.seq, elementMetadata(promoters.seq)$probe)
```

Next, we can count the number of reads falling into our promoters:

```
> chipSetRaw.seq <- summarizeReads(mappedReads, promoters.seq, summarize="add")
> chipSetRaw.seq

class: ChIPseqSet
dim: 3502 2
metadata(0):
assays(1): chipVals
rownames(3502): TSS1000 TSS10001 ... TSS9998 TSS9999
rowData names(0):
colnames(2): CEBPA_WT_1 CEBPA_KO_1
colData names(1): totalCount

> head(chipVals(chipSetRaw.seq))

         CEBPA_WT_1 CEBPA_KO_1
TSS1000           0          0
TSS10001          0          0
TSS10003          0          0
TSS10004          0          0
TSS10007          0          0
TSS10013          0          0
```

From now on, we do not distinguish between microarray and RNA-seq any more. `eSet` can be substituted by `eSet.ser` and `chipSetRaw` by `chipSetRaw.seq`. In the following, the microarray data is used, since the RNA-seq data was not obtained from the same samples as the ChIP-seq data (actually, not even the same organism).

3 ChIP-seq data normalization

It may be necessary to normalize ChIP-seq data due to different experimental conditions during ChIP.

```
> chipSet <- normalize(chipSetRaw, method="quantile")
```

In addition to quantile normalization, other methods like the method presented by [**?**] are available.

```
> par(mfrow=c(1,2))
> plot(chipVals(chipSetRaw)[,1], chipVals(chipSetRaw)[,2],
      xlim=c(1,600), ylim=c(1,600), main="Raw")
> plot(chipVals(chipSet)[,1], chipVals(chipSet)[,2],
      xlim=c(1,600), ylim=c(1,600), main="Quantile")
```

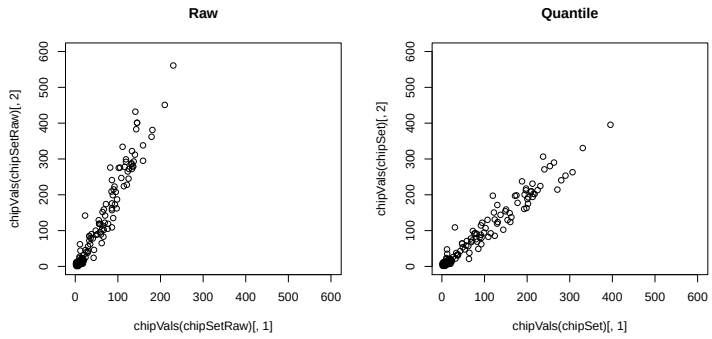

Figure 1: Raw and quantile normalized ChIP-seq data.

4 Data integration

In order to integrate both data types, a correlation score Z (motivated by the work of [**?**]) can be calculated by multiplying the standardized difference of gene expression values with the standardized difference of ChIP-seq values. Prior to this, pheno type information must be added to the `chipSet` object.

```
> eSet$CEBPA

[1] wt wt ko ko
Levels: ko wt

> colnames(chipSet)

[1] "CEBPA_WT_1" "CEBPA_KO_1"

> chipSet$CEBPA <- factor(c("wt", "ko"))
> colData(chipSet)

DataFrame with 2 rows and 2 columns
           totalCount    CEBPA
            <integer> <factor>
CEBPA_WT_1       8687       wt
CEBPA_KO_1      17122       ko

> intData <- integrateData(eSet, chipSet,
     factor="CEBPA", reference="wt")
> head(intData)

          expr_ko  expr_wt chipseq_ko chipseq_wt          z
10354832 8.864536 8.392561      193.0      202.5 -0.8048761
10359770 7.161367 7.305733      213.0      224.5  0.2980229
10355974 7.956849 7.850496      214.5      271.0 -1.0786664
10348378 5.384252 5.339577       49.0       85.5 -0.2927146
10353775 4.780612 4.700385       15.0       13.5  0.0216021
10352827 6.175612 5.873558        8.5        8.5  0.0000000
```

5 Classification by mixture models

5.1 Maximum likelihood approach

We now fit a mixture model to the correlation score Z. The model consists of two normal components with fixed $\mu = 0$. These two components should capture Z values close to zero, i.e. genes that show no differences between wild type and knock-out in at least one of the two data sets. The positive (negative) Z scores are represented by a (mirrored) exponential component. Parameters are estimated using the EM-algorithm as implemented in the method `mlMixModel`.

```
> mlmm = mlMixModel(intData[,"z"],
     normNull=c(2, 3), expNeg=1, expPos=4,
     sdNormNullInit=c(0.5, 1), rateExpNegInit=0.5, rateExpPosInit=0.5,
     pi=rep(1/4, 4))

> mlmm

MixModel object
    Number of data points:  180
    Number of components:   4
        1: ExpNeg
             rate = 1.532987
           weight pi = 0.2219707
           classified data points: 30
        2: NormNull
             mean = 0
             sd = 0.01644812
           weight pi = 0.2154126
           classified data points: 48
        3: NormNull
             mean = 0
             sd = 0.1213587
           weight pi = 0.3526906
           classified data points: 70
        4: ExpPos
             rate = 0.6931467
           weight pi = 0.2099261
           classified data points: 32
```

The method returns an object of class *MixModelML*, a subclass of *MixModel*. We now plot the model fit and the classification results:

```
> par(mfrow=c(1,2))
> plotComponents(mlmm, xlim=c(-2, 2), ylim=c(0, 3))
> plotClassification(mlmm)
```

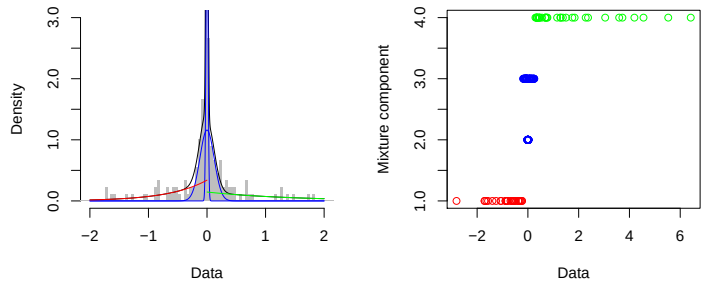

Figure 2: Model fit and classification results of the maximum likelihood approach.

5.2 Bayesian approach

Alternatively, an Bayesian approach can be used.

```
> set.seed(1515)
> bayesmm = bayesMixModel(intData[,"z"],
     normNull=c(2, 3), expNeg=1, expPos=4,
     sdNormNullInit=c(0.5, 1), rateExpNegInit=0.5, rateExpPosInit=0.5,
     shapeNorm0=c(10, 10), scaleNorm0=c(10, 10), shapeExpNeg0=0.01,
     scaleExpNeg0=0.01, shapeExpPos0=0.01, scaleExpPos0=0.01,
     pi=rep(1/4, 4), sdAlpha=1, itb=2000, nmc=8000, thin=5)
```

`bayesMixModel` returns an object of class *MixModelBayes*, which is also a subclass of *MixModel*.

```
> bayesmm
```

```
MixModel object
    Number of data points:  180
    Number of components:   4
        1: ExpNeg
             rate = 7.946674e-05
           weight pi = 0.000248503
           classified data points: 0
        2: NormNull
             mean = 0
             sd = 0.08108583
```

```
   weight pi = 0.5518981
   classified data points: 111
3: NormNull
     mean = 0
     sd = 0.7907854
   weight pi = 0.3967847
   classified data points: 61
4: ExpPos
     rate = 0.06959121
   weight pi = 0.05106871
   classified data points: 8
```

The same methods for plotting the model fit and classification can be applied.

```
> par(mfrow=c(1,2))
> plotComponents(bayesmm, xlim=c(-2, 2), ylim=c(0, 3))
> plotClassification(bayesmm, method="mode")
```

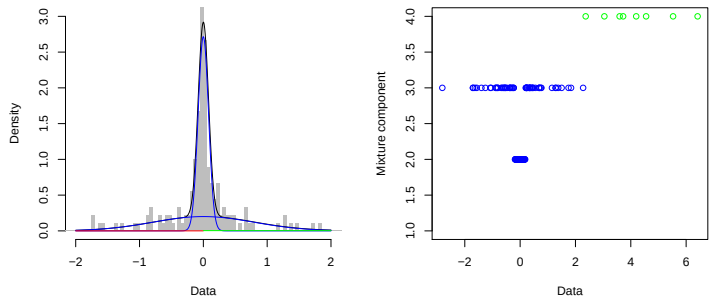

Figure 3: Model fit and classification results of the Bayesian approach.

Note, that the parameters 'burn in' (`itb`) and 'number of iterations' (`nmc`) have to be chosen carefully. The method `plotChains` should be used to assess the convergence of the markov chains for each parameter. The settings here lead to a short runtime, but are unsuitable for real applications.

Both models tend to classify more genes to the positive component (component 4) than to the negative one (component 1):

```
> table(classification(mlmm, method="maxDens"),
      classification(bayesmm, method="mode"))

    2  3  4
1   0 30  0
```

```
  2 48  0  0
  3 63  7  0
  4  0 24  8
```

This is in line with the fact that H3K4me3 occurs in the promoters of active genes. Since each z corresponds to a probeset (and so to at least one transcript), the corresponding microarray annotation packages can be used to obtain e.g. the gene symbols of all positivly classified z scores.

```
> posProbes <- rownames(intData)[classification(bayesmm, method="mode") == 4]
> library("mogene10sttranscriptcluster.db")
> unlist(mget(posProbes, mogene10sttranscriptclusterSYMBOL))
```