The genomic STate ANnotation package

Benedikt Zacher[1,2,*], Julia Ertl[1], Julien Gagneur[1], Achim Tresch[1,2,3]

[1] Gene Center and Department of Biochemistry, LMU, Munich, Germany
[2] Institute for Genetics, University of Cologne, Cologne, Germany
[3] Max Planck Institute for Plant Breeding Research, Cologne, Germany

[*]zacher (at) genzentrum.lmu.de

May 30, 2016

If you use *STAN* in published research, please cite:

Zacher, B. and Lidschreiber, M. and Cramer, P. and Gagneur, J. and Tresch, A. (2014): **Annotation of genomics data using bidirectional hidden Markov models unveils variations in Pol II transcription cycle** *Mol. Syst. Biol.* **10**:768

Contents

1 Quick start

```
## Loading library and data
library(STAN)
data(trainRegions)
data(pilot.hg19)

## Model initialization
hmm_nb = initHMM(trainRegions[1:3], nStates=10, "NegativeBinomial")

## Model fitting
hmm_fitted_nb = fitHMM(trainRegions[1:3], hmm_nb, maxIters=10)

## Calculate state path
viterbi_nb = getViterbi(hmm_fitted_nb, trainRegions[1:3])

## Convert state path to GRanges object
viterbi_nb_gm12878 = viterbi2GRanges(viterbi_nb, pilot.hg19, 200)
```

2 Introduction

Genome segmentation with hidden Markov models has become a useful tool to annotate genomic elements, such as promoters and enhancers by data integration. *STAN* (genomic **ST**ate **AN**notation) implements (bidirectional) Hidden Markov Models (HMMs) using a variety of different probability distributions, which can be used to model a wide range of current genomic data:

- Multivariate gaussian: e.g. conutinuous microarray and transformed sequencing data.
- Poisson: e.g. discrete count data from sequencing experiments.
- (Zero-Inflated) Negative Binomial: e.g. discrete count data from sequencing experiments.
- Poisson Log-Normal: e.g. discrete count data from sequencing experiments.
- Negative Multinomial: e.g. discrete count data from sequencing experiments.
- Multinomial: e.g. methylation rates from bisulfite sequencing (in this case it reduces to a Binomial) or binned nuceotide frequencies.
- Bernoulli: Initially proposed by [**?**] to model presence/absence of chromatin marks (another example: transcription factor binding).
- Nucleotide distribution: e.g. nucleotide frequencies in the DNA sequence.

The use of these distributions enables integrating a wide range of genomic data types (e.g. continuous, discrete, binary) to *de novo* learn and annotate the genome into a given number of 'genomic states'. The 'genomic states' may for instance reflect distinct genome-associated protein complexes or describe recurring patterns of chromatin features, i.e. the 'chromatin state'. Unlike other tools, *STAN* also allows for the integration of strand-specific (e.g. RNA) and non-strand-specific data (e.g. ChIP) [**?**]. In this vignette, we illustrate the use of *STAN* by inferring 'chromatin states' from a small example data set of two Roadmap Epigenomics cell lines. Moreover, we show how to use strand-specific RNA expression with non-strand-specific ChIP data to infer 'transcription states' in yeast. Before getting started the package needs to be loaded:

```
library(STAN)
```

3 Genomic state annotation of Roadmap Epigenomics Sequencing data

The data (or observations) provided to *STAN* may consist of one or more observation sequences (e.g. chromosomes), which are contained in a list of (position x experiment) matrices. `trainRegions` is a list containing one three ENCODE pilot regions (stored in `pilot.hg19` as *GRanges* object) with data for two cell types (K562: E123, Gm12878: E116) from the Roadmap Epigenomics project. The data set contains ChIP-Seq experiments of seven histone modifications (H3K4me1, H3K4me3, H3K36me3, H3K27me3, H3K27ac and H3K9ac), as well as DNase-Seq and genomic input.

```
data(trainRegions)
names(trainRegions)
## [1] "E116.chr1.ENr231"  "E116.chr10.ENr114" "E116.chr11.ENm011" "E123.chr1.ENr231"
## [5] "E123.chr10.ENr114" "E123.chr11.ENm011"
str(trainRegions[c(1,4)])
## List of 2
##  $ E116.chr1.ENr231: num [1:2500, 1:9] 2 4 2 0 1 1 4 4 2 14 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : NULL
##   .. ..$ : chr [1:9] "H3K4me1" "H3K4me3" "H3K36me3" "H3K27me3" ...
##  $ E123.chr1.ENr231: num [1:2500, 1:9] 2 1 5 0 1 3 8 8 7 12 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : NULL
##   .. ..$ : chr [1:9] "H3K4me1" "H3K4me3" "H3K36me3" "H3K27me3" ...
```

The genomic regions for each cell type in `trainRegions` are stored as a GRanges object in `pilot.hg19`:

```
data(pilot.hg19)
pilot.hg19
## GRanges object with 3 ranges and 1 metadata column:
##       seqnames                ranges strand |        name
##          <Rle>             <IRanges>  <Rle> | <character>
##   [1]     chr1 [151158001, 151658000]      * |      ENr231
##   [2]    chr10 [ 55483801,  55983800]      * |      ENr114
##   [3]    chr11 [  1743401,   2349400]      * |      ENm011
##   -------
##   seqinfo: 21 sequences from an unspecified genome; no seqlengths
```

Before model fitting, we calculate size factors to correct for the different different sequencing depths between cell lines.

```
celltypes = list("E123"=grep("E123", names(trainRegions)),
       "E116"=grep("E116", names(trainRegions)))
sizeFactors = getSizeFactors(trainRegions, celltypes)
sizeFactors
##       H3K4me1   H3K4me3  H3K36me3 H3K27me3   H3K9me3  H3K27ac    H3K9ac     DNase
## E123 1.055900 0.9104679 0.9397551 0.946867 1.2351436 1.112482 1.2326048 0.9252687
## E123 1.055900 0.9104679 0.9397551 0.946867 1.2351436 1.112482 1.2326048 0.9252687
## E123 1.055900 0.9104679 0.9397551 0.946867 1.2351436 1.112482 1.2326048 0.9252687
## E116 0.949721 1.1090610 1.0684983 1.059451 0.8400696 0.908175 0.8412481 1.0878636
## E116 0.949721 1.1090610 1.0684983 1.059451 0.8400696 0.908175 0.8412481 1.0878636
## E116 0.949721 1.1090610 1.0684983 1.059451 0.8400696 0.908175 0.8412481 1.0878636
##          Input
## E123 1.2844840
## E123 1.2844840
```

```
## E123 1.2844840
## E116 0.8186808
## E116 0.8186808
## E116 0.8186808
```

Genome segmentation is carried out in *STAN* using three functions: `initHMM`, `fitHMM` and `getViterbi`. `initHMM` initializes a model with `nStates` states for a given probability/emission distribution, which we set to 'PoissonLogNormal' in this example. `fitHMM` then optimizes model parameters using the Expectation-Maximization algorithm. Model parameters can be accessed with the `EmissionParams` function. Note that in this example, we set the maximal number of iteration to 10 in this case for speed reason. To ensure convergence this number should be higher in real world applications. After HMM fitting, the state annotation is calculated using the `getViterbi` function.

```
nStates = 10
hmm_poilog = initHMM(trainRegions, nStates, "PoissonLogNormal", sizeFactors)
hmm_fitted_poilog = fitHMM(trainRegions, hmm_poilog, sizeFactors=sizeFactors, maxIters=10)
viterbi_poilog = getViterbi(hmm_fitted_poilog, trainRegions, sizeFactors)
str(viterbi_poilog)
## List of 6
##  $ E116.chr1.ENr231 : Factor w/ 10 levels "1","2","3","4",..: 1 1 1 1 1 1 2 2 2 4 ...
##  $ E116.chr10.ENr114: Factor w/ 10 levels "1","2","3","4",..: 10 10 10 10 10 10 10 10 10 10 ...
##  $ E116.chr11.ENm011: Factor w/ 10 levels "1","2","3","4",..: 10 10 3 3 10 10 10 10 10 10 ...
##  $ E123.chr1.ENr231 : Factor w/ 10 levels "1","2","3","4",..: 1 1 1 1 1 2 2 2 2 2 ...
##  $ E123.chr10.ENr114: Factor w/ 10 levels "1","2","3","4",..: 10 10 10 10 10 10 10 10 10 10 ...
##  $ E123.chr11.ENm011: Factor w/ 10 levels "1","2","3","4",..: 3 3 3 3 3 3 3 3 3 3 ...
```

In order to ease the use of other genomic applications and Bioconductor packages, the viterbi path can be converted into a *GRanges* object.

```
viterbi_poilog_gm12878 = viterbi2GRanges(viterbi_poilog[1:3], regions=pilot.hg19, binSize=200)
viterbi_poilog_gm12878
## GRanges object with 802 ranges and 1 metadata column:
##         seqnames                ranges strand |        name
##            <Rle>             <IRanges>  <Rle> | <character>
##     [1]     chr1 [151158001, 151159201]      * |           1
##     [2]     chr1 [151159201, 151159801]      * |           2
##     [3]     chr1 [151159801, 151160801]      * |           4
##     [4]     chr1 [151160801, 151161001]      * |           9
##     [5]     chr1 [151161001, 151161601]      * |           8
##     ...      ...                   ...    ... .         ...
##   [798]    chr11     [2340001, 2341001]      * |          10
##   [799]    chr11     [2341001, 2343001]      * |           3
##   [800]    chr11     [2343001, 2346401]      * |          10
##   [801]    chr11     [2346401, 2348801]      * |           3
##   [802]    chr11     [2348801, 2349401]      * |          10
##   -------
##   seqinfo: 3 sequences from an unspecified genome; no seqlengths
```

Before giving some more details about further analysis and visualization of the models we repeat above segementations using the 'NegativeBinomial' emission functions.

```
hmm_nb = initHMM(trainRegions, nStates, "NegativeBinomial", sizeFactors)
hmm_fitted_nb = fitHMM(trainRegions, hmm_nb, sizeFactors=sizeFactors, maxIters=10)
viterbi_nb = getViterbi(hmm_fitted_nb, trainRegions, sizeFactors=sizeFactors)
```

```
viterbi_nb_gm12878 = viterbi2GRanges(viterbi_nb[1:3], pilot.hg19, 200)
```

In order to assign biologically meaningful roles to the inferred states we calculate the mean number of reads per 200 base pair bin for both segmentations.

```
avg_cov_nb = getAvgSignal(viterbi_nb, trainRegions)
avg_cov_poilog = getAvgSignal(viterbi_poilog, trainRegions)
```

These are then plotted using the `heatmap.2` function (see Figure **??**).

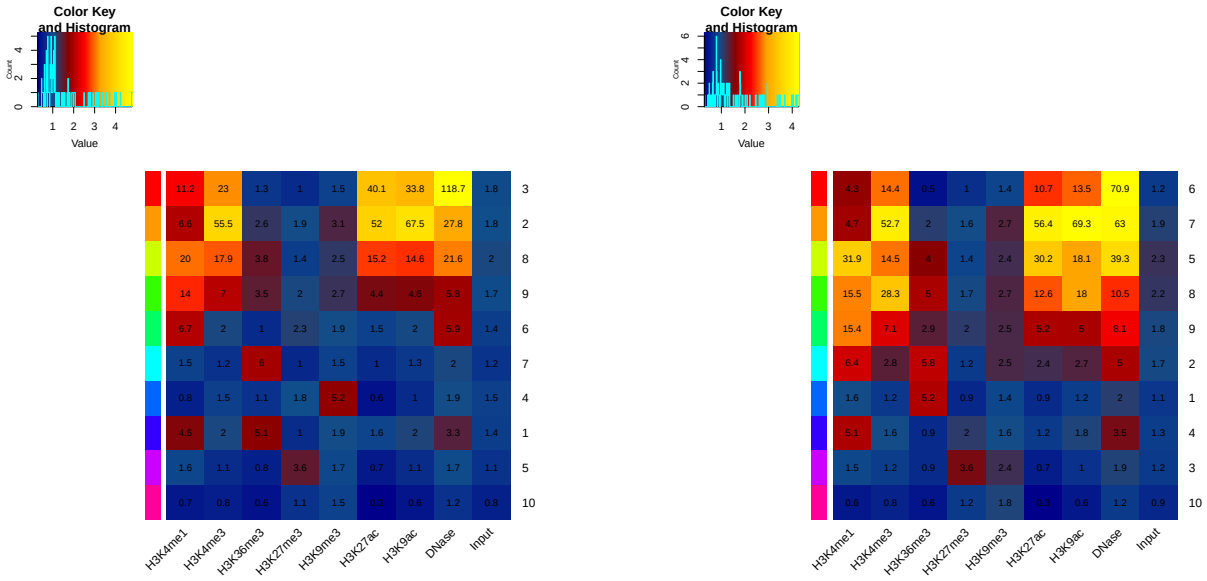
```
## specify color palette
library(gplots)
heat = c("dark blue", "dodgerblue4", "darkred", "red", "orange", "gold", "yellow")
colfct = colorRampPalette(heat)
colpal_statemeans = colfct(200)

## define state order and colors
ord_nb = order(apply(avg_cov_nb,1,max), decreasing=TRUE)
statecols_nb = rainbow(nStates)
names(statecols_nb) = ord_nb
heatmap.2(log(avg_cov_nb+1)[as.character(ord_nb),], margins=c(8,7), srtCol=45,
        RowSideColors=statecols_nb[as.character(ord_nb)], dendrogram="none",
        Rowv=FALSE, Colv=FALSE, col=colpal_statemeans, trace="none",
        cellnote=round(avg_cov_nb,1)[as.character(ord_nb),], notecol="black")
## define state order and colors
ord_poilog = order(apply(avg_cov_poilog,1,max), decreasing=TRUE)
statecols_poilog = rainbow(nStates)
names(statecols_poilog) = ord_poilog
heatmap.2(log(avg_cov_poilog+1)[as.character(ord_poilog),], margins=c(8,7), srtCol=45,
        RowSideColors=statecols_poilog[as.character(ord_poilog)], dendrogram="none",
        Rowv=FALSE, Colv=FALSE, col=colpal_statemeans, trace="none",
        cellnote=round(avg_cov_poilog,1)[as.character(ord_poilog),], notecol="black")
```

In order to visualize both *STAN* segmentations, we convert the viterbi paths and the data to *Gviz* objects.

```
library(Gviz)
from = start(pilot.hg19)[3]
to = from+300000
gvizViterbi_nb = viterbi2Gviz(viterbi_nb_gm12878, "chr11", "hg19", from, to, statecols_nb)
gvizViterbi_poilog = viterbi2Gviz(viterbi_poilog_gm12878, "chr11", "hg19", from, to,
        statecols_poilog)
gvizData = data2Gviz(trainRegions[3], pilot.hg19[3], 200, "hg19", col="black")
```

Then, we use the `plotTracks` function to plot everything (see Figure **??**).

```
gaxis = GenomeAxisTrack()
data(ucscGenes)
mySize = c(1,rep(1.2,9), 0.5,0.5,3)
plotTracks(c(list(gaxis), gvizData,gvizViterbi_nb,gvizViterbi_poilog,ucscGenes["chr11"]),
        from=from, to=to, showFeatureId=FALSE, featureAnnotation="id", fontcolor.feature="black",
        cex.feature=0.7, background.title="darkgrey", lwd=2, sizes=mySize)
```



(a) (b)

Figure 1: Mean read counts of the (a) 'NegativeBinomial' and (b) 'PoissonLogNormal' state annotation

Modeling Sequencing data using other emission functions

In this section we illustrate the use of other distributions to annotate the the Roadmap Epigenomics example data set, namely the 'Poisson', 'NegativeMultinomial', 'Gaussian' and 'Bernoulli' models. The 'Poisson' model is an obvious choice when dealing with count data. However since the variance of the Poisson is equal to its mean it might not be an ideal choice for modeling Sequencing experiments, which have been shown to be overdispersed [**?**].

```
hmm_pois = initHMM(trainRegions, nStates, "Poisson")
hmm_fitted_pois = fitHMM(trainRegions, hmm_pois, maxIters=10)
viterbi_pois = getViterbi(hmm_fitted_pois, trainRegions)
```

The 'NegativeMultinomial' distribution for genome segmentation with HMMs was first proposed in the EpicSeg model [**?**]. The Negative Multinomial can be understood as a Multinomial distribution, where its overdispersion of is modeled by a Negative Binomial distribution. However, this assumes a shared overdispersion across data tracks within a state as opposed to the 'NegativeBinomial' and 'PoissonLogNormal' models which model the variance for each state and data track separately. In order to use the 'NegativeMultinomial' in *STAN* an additional data track - the sum of counts - for each bin needs to be added to the data. Internally the 'NegativeMultinomial' is modeled as a product of a 'NegativeBinomial' and a 'Multinomial' emission (see section 'Combining different emission functions' for further details):

```
simData_nmn = lapply(trainRegions, function(x) cbind(apply(x,1,sum), x))
hmm_nmn = initHMM(simData_nmn, nStates, "NegativeMultinomial")
hmm_fitted_nmn = fitHMM(simData_nmn, hmm_nmn, maxIters=10)
viterbi_nmn = getViterbi(hmm_fitted_nmn, simData_nmn)
```

In order to model the data using Gaussian distributions, it needs to be log-transformed and smoothed. This approach is implementd in Segway, a method used by the ENCODE Consortium for chromatin state annotation [**?**]. However, to overcome singularity of the (diagonal) covariance matrix due to the zero-inflated distribution of the transformed read counts, it uses a shared variance over states for each data track. To use gaussian distributions with Sequencing data in *STAN*, we transform the data (with the hyperbole sine function [**?**]) and model it using the emission 'IndependentGaussian'

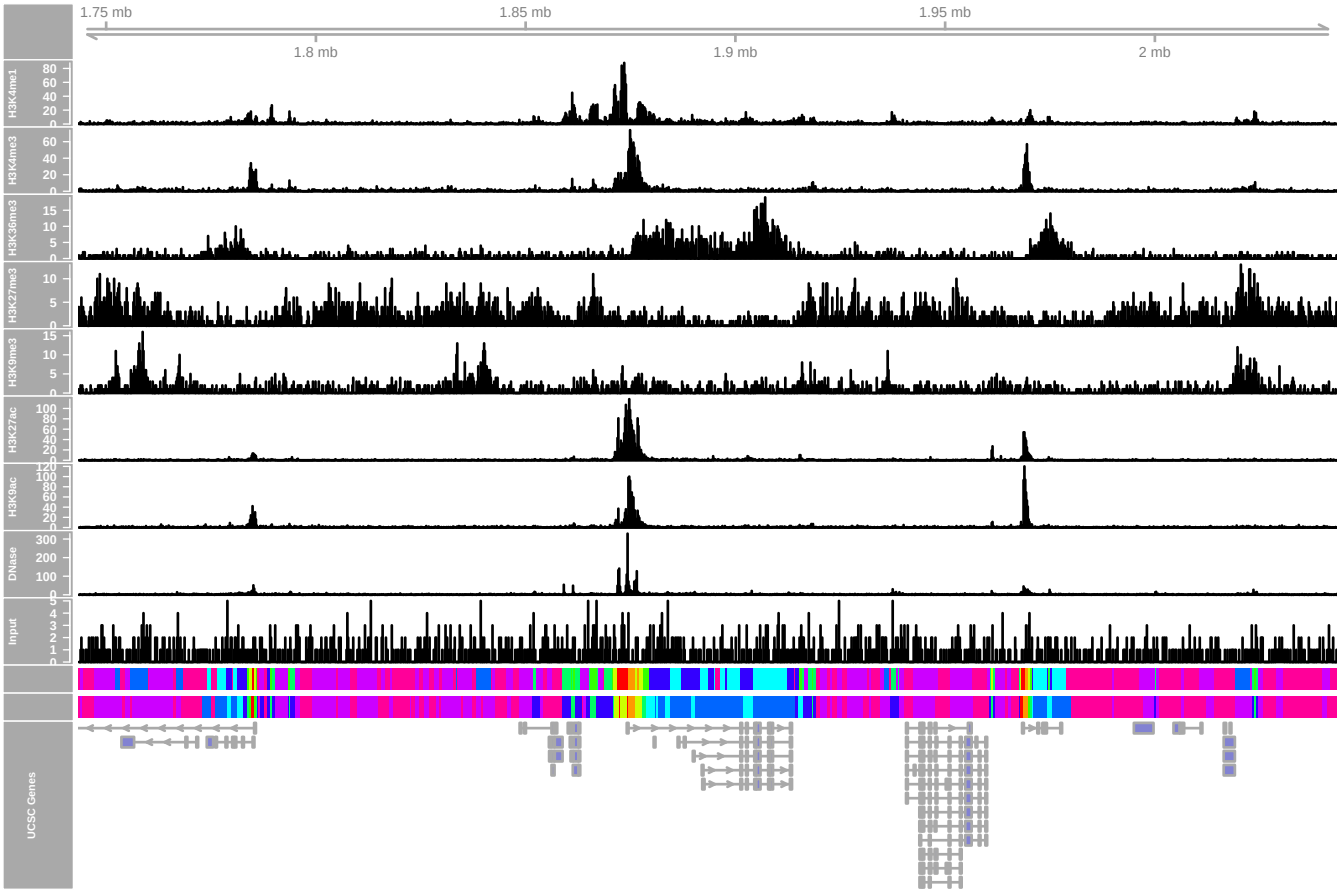


Figure 2: Genome Browser showing the 10 data tracks used for model learning together with the 'Negativebinomial' (top) and 'PoissonLogNormal' (bottom) segmentations and known UCSC gene annotations.

with a shared covariance, i.e. `sharedCov=TRUE`.

```
trainRegions_smooth = lapply(trainRegions, function(x)
            apply(log(x+sqrt(x^2+1)), 2, runningMean, 2))
hmm_gauss = initHMM(trainRegions_smooth, nStates, "IndependentGaussian", sharedCov=TRUE)
hmm_fitted_gauss = fitHMM(trainRegions_smooth, hmm_gauss, maxIters=10)
viterbi_gauss = getViterbi(hmm_fitted_gauss, trainRegions_smooth)
```

Another approach was proposed in ChromHMM, which models binarized data using an independent Bernoulli model [**?**]. Note, that the performance of the model highly depends on the non-trivial choice of a proper cutoff and quantitative information is lost. The latter is especially important when predicting promoters and enhancers since these elements are both marked H3K4me1 and H3K4me3, but at different ratios. The function `binarizeData` binarizes the data using the default approach by ChromHMM [**?**]. The model can then be fit by specifying the 'Bernoulli' model. Note however, that initialization and model fitting are carried out differently than in the ChromHMM implementation. In particular *STAN* uses the EM algorithm while ChromHMM uses online EM. For details on the initialization, please see the `initHMM` manual.

```
trainRegions_binary = binarizeData(trainRegions)
hmm_ber = initHMM(trainRegions_binary, nStates, "Bernoulli")
hmm_fitted_ber = fitHMM(trainRegions_binary, hmm_ber, maxIters=10)
```

```
viterbi_ber = getViterbi(hmm_fitted_ber, trainRegions_binary)
```

We calculate the mean read coverage for each method and segmentation:

```
avg_cov_gauss = getAvgSignal(viterbi_gauss, trainRegions)
avg_cov_nmn = getAvgSignal(viterbi_nmn, trainRegions)
avg_cov_ber = getAvgSignal(viterbi_ber, trainRegions)
avg_cov_pois = getAvgSignal(viterbi_pois, trainRegions)
```

These are again plotted using the `heatmap.2` function (see Figure **??**).

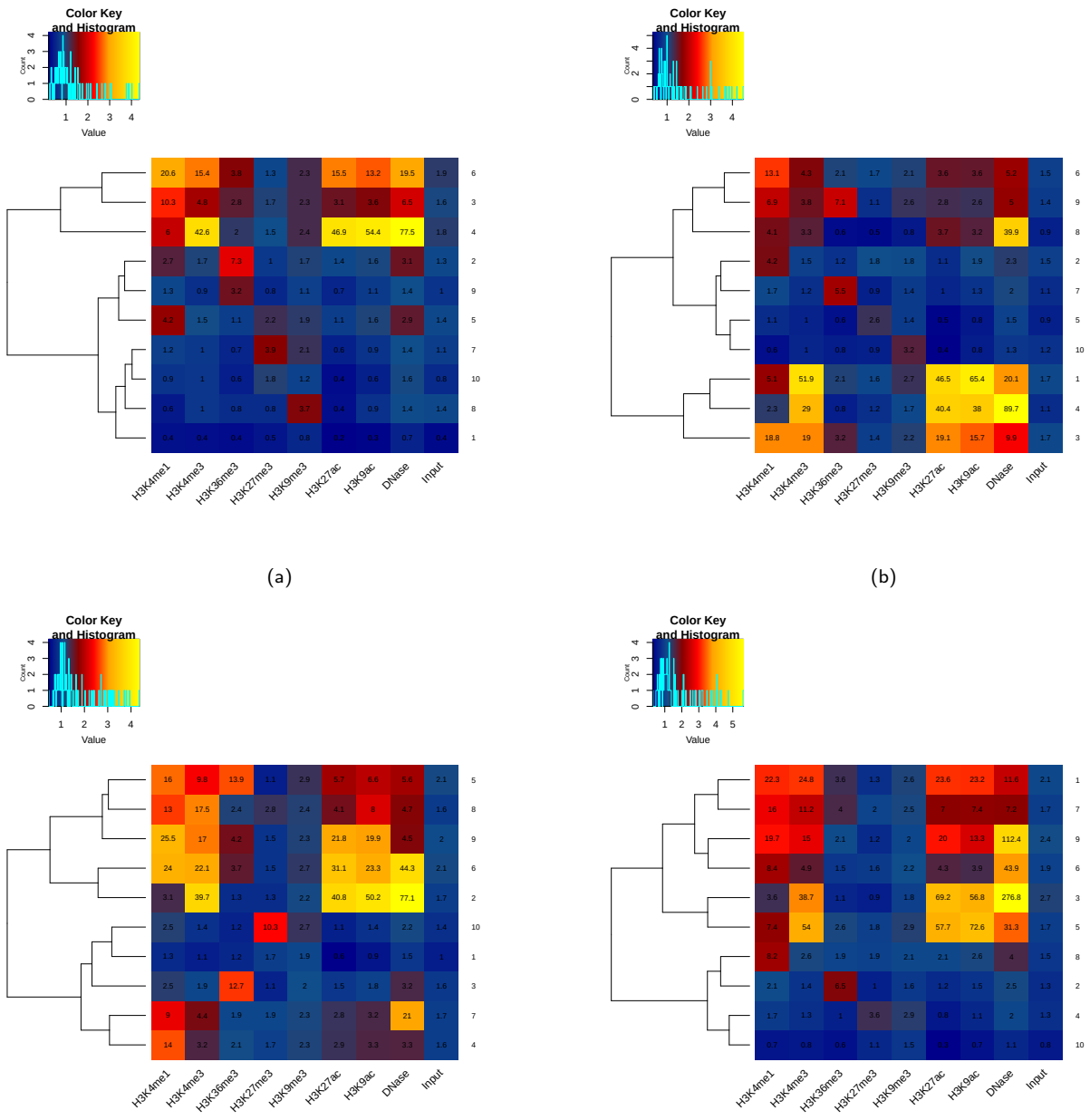
```
heatmap.2(log(avg_cov_gauss+1), margins=c(8,7),srtCol=45, dendrogram="row", Rowv=TRUE,
        Colv=FALSE, col=colpal_statemeans, trace="none", notecex=0.7, cexRow=0.75, cexCol=1,
        cellnote=round(avg_cov_gauss,1), notecol="black")
heatmap.2(log(avg_cov_nmn+1), margins=c(8,7),srtCol=45, dendrogram="row", Rowv=TRUE,
        Colv=FALSE, col=colpal_statemeans, trace="none", notecex=0.7, cexRow=0.75, cexCol=1,
        cellnote=round(avg_cov_nmn,1), notecol="black")
heatmap.2(log(avg_cov_ber+1), margins=c(8,7),srtCol=45, dendrogram="row", Rowv=TRUE,
        Colv=FALSE, col=colpal_statemeans, trace="none", notecex=0.7, cexRow=0.75, cexCol=1,
        cellnote=round(avg_cov_ber,1), notecol="black")
heatmap.2(log(avg_cov_pois+1), margins=c(8,7),srtCol=45, dendrogram="row", Rowv=TRUE,
        Colv=FALSE, col=colpal_statemeans, trace="none", notecex=0.7, cexRow=0.75, cexCol=1,
        cellnote=round(avg_cov_pois,1), notecol="black")
```

4 Integrating strand-specific and non-strand-specific data with STAN

STAN also allows for the integration of strand-specific (e.g. RNA) and non-strand-specific data (e.g. ChIP). This is donw using bidirectional hidden Markov models (bdHMMs) which were proposed in [**?**]. A bdHMM models a directed process using the concept of twin states, where each genomic state is split up into a pair of twin states, one for each direction (e.g. sense and antisense in context of transcription). Those twin state pairs are identical in terms of their emissions (i.e. they model the same genomic state). Currently the following models are available for bdHMMs: 'IndependentGaussian', 'Gaussian', 'NegativeBinomial', 'ZINegativeBinomial' and 'PoissonLogNormal'. We now illustrate the use of bdHMMs in *STAN* at an example data set of yeast transcription factors measured by ChIP-chip and RNA expression measured with a tiling array which was used to model the transcription cycle as a sequence of 'transcription states' in [**?**].
The `initBdHMM` function is used to initialize a bdHMM with 6 twin states. Note that the overall number of states in the bdHMM is 12 (6 identical twin state pairs). `dirobs` defines the directionality (or strand-specificity) of the data tracks. In `dirobs`, the first 10 data tracks are non-strand-specific ChIP-chip measurments, indicated by '0' and data track 11 and 12 are strand-specific RNA expression measurements, indicated by '1'. Note that strand-specific data tracks must be labeled as increasing pairs of integers. Thus and additional strand-specific data track pair would be labeled as a pair of '2'. Model fitting and calculation of the state annotation are carried out as for standard HMMs:

```
data(yeastTF_databychrom_ex)
nStates = 6
dirobs = as.integer(c(rep(0,10), 1, 1))
bdhmm_gauss = initBdHMM(yeastTF_databychrom_ex, nStates, "Gaussian", directedObs=dirobs)
bdhmm_fitted_gauss = fitHMM(yeastTF_databychrom_ex, bdhmm_gauss)
viterbi_bdhmm_gauss = getViterbi(bdhmm_fitted_gauss, yeastTF_databychrom_ex)
```

We plot the means of the multivariate gaussian distrbutions for each state (see Figure **??**):



(a) (b)

(c) (d)

Figure 3: Mean read counts of the (a) 'IndependentGaussian' (b) 'NegativeMultinomial' (c) 'Bernoulli' and (d) 'Poisson' state annotation.

```
statecols_yeast = rep(rainbow(nStates), 2)
names(statecols_yeast) = StateNames(bdhmm_fitted_gauss)
means_fitted = EmissionParams(bdhmm_fitted_gauss)$mu
heatmap.2(means_fitted, col=colpal_statemeans,
        RowSideColors=statecols_yeast[rownames(means_fitted)],
        trace="none", cexCol=0.9, cexRow=0.9,
        cellnote=round(means_fitted,1), notecol="black", dendrogram="row",
```

```
        Rowv=TRUE, Colv=FALSE, notecex=0.9)
```

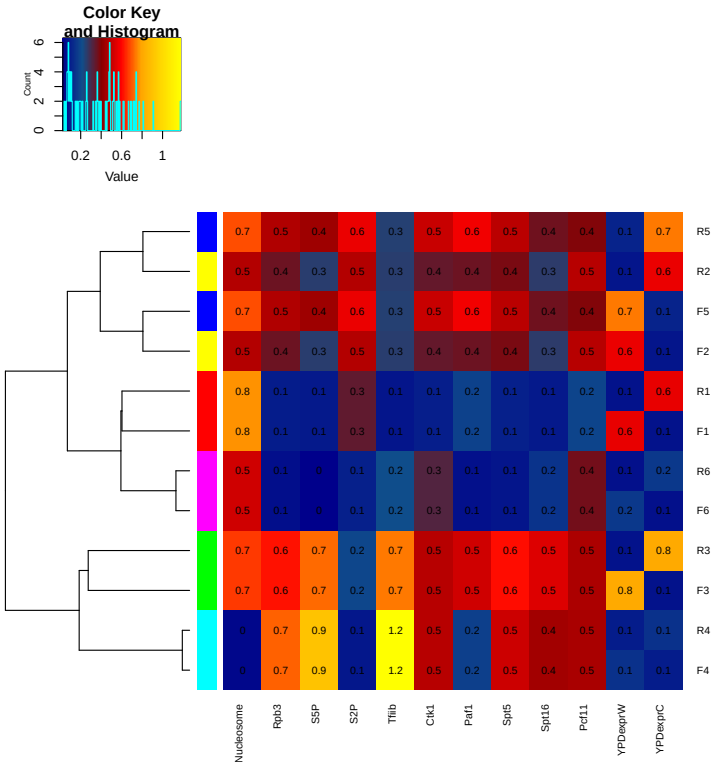


Figure 4: Mean signal 6 bdHMM twin state pairs. 'F' and 'R' indicate forward and reverse direction of state pairs.

We convert the viterbi path into a `GRanges` object. Note that the directionaliy of bdHMM states is indicated by 'F' (forward) and 'R' (reverse).

```
yeastGRanges = GRanges(IRanges(start=1214616, end=1225008), seqnames="chrIV")
names(viterbi_bdhmm_gauss) = "chrIV"
viterbi_bdhmm_gauss_gr = viterbi2GRanges(viterbi_bdhmm_gauss, yeastGRanges, 8)
viterbi_bdhmm_gauss_gr
## GRanges object with 68 ranges and 1 metadata column:
##        seqnames            ranges strand |        name
##           <Rle>         <IRanges>  <Rle> | <character>
##    [1]    chrIV [1214616, 1215016]      * |          F5
##    [2]    chrIV [1215016, 1215088]      * |          R5
##    [3]    chrIV [1215088, 1215224]      * |          R1
##    [4]    chrIV [1215224, 1215280]      * |          R2
##    [5]    chrIV [1215280, 1215448]      * |          R1
##    ...      ...                 ...    ... .         ...
##   [64]    chrIV [1224616, 1224632]      * |          F6
##   [65]    chrIV [1224632, 1224696]      * |          R6
##   [66]    chrIV [1224696, 1224752]      * |          F6
##   [67]    chrIV [1224752, 1224904]      * |          F3
##   [68]    chrIV [1224904, 1225008]      * |          F2
##   -------
##   seqinfo: 1 sequence from an unspecified genome; no seqlengths
```

Next, we visualize the data, state annotation and together with SGD genes using *Gviz* (see Figure **??**):

```
chr = "chrIV"
gen = "sacCer3"
gtrack <- GenomeAxisTrack()

from=1217060
to=1225000
forward_segments = grep("F", viterbi_bdhmm_gauss_gr$name)
reverse_segments = grep("R", viterbi_bdhmm_gauss_gr$name)
gvizViterbi_yeast = viterbi2Gviz(viterbi_bdhmm_gauss_gr[forward_segments],
        "chrIV", "sacCer3", from, to, statecols_yeast)
gvizViterbi_yeast2 = viterbi2Gviz(viterbi_bdhmm_gauss_gr[reverse_segments],
        "chrIV", "sacCer3", from, to, statecols_yeast)

gvizData_yeast = data2Gviz(yeastTF_databychrom_ex, yeastGRanges, 8, "sacCer3", col="black")
gaxis = GenomeAxisTrack()
data(yeastTF_SGDGenes)
mySize = c(1,rep(1,12), 0.5,0.5,3)

plotTracks(c(list(gaxis), gvizData_yeast,gvizViterbi_yeast,gvizViterbi_yeast2,
        list(yeastTF_SGDGenes)), cex.feature=0.7, background.title="darkgrey", lwd=2,
        sizes=mySize, from=from, to=to, showFeatureId=FALSE, featureAnnotation="id",
        fontcolor.feature="black", cex.feature=0.7, background.title="darkgrey",
        showId=TRUE)
```

5 Concluding Remarks

This vignette was generated using the following package versions:

- R version 3.3.0 (2016-05-03), `x86_64-apple-darwin13.4.0`
- Locale: `C/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8`
- Base packages: base, datasets, grDevices, graphics, grid, methods, parallel, stats, stats4, utils
- Other packages: BiocGenerics 0.18.0, GenomeInfoDb 1.8.2, GenomicRanges 1.24.0, Gviz 1.16.1, IRanges 2.6.0, S4Vectors 0.10.1, STAN 2.0.3, gplots 3.0.1, knitr 1.13, poilog 0.4
- Loaded via a namespace (and not attached): AnnotationDbi 1.34.3, AnnotationHub 2.4.2, BSgenome 1.40.0, Biobase 2.32.0, BiocInstaller 1.22.2, BiocParallel 1.6.2, BiocStyle 2.0.2, Biostrings 2.40.1, DBI 0.4-1, Formula 1.2-1, GenomicAlignments 1.8.0, GenomicFeatures 1.24.2, Hmisc 3.17-4, KernSmooth 2.23-15, Matrix 1.2-6, R6 2.1.2, RColorBrewer 1.1-2, RCurl 1.95-4.8, RSQLite 1.0.0, Rcpp 0.12.5, Rsamtools 1.24.0, Rsolnp 1.16, SummarizedExperiment 1.2.2, VariantAnnotation 1.18.1, XML 3.98-1.4, XVector 0.12.0, acepack 1.3-3.3, biomaRt 2.28.0, biovizBase 1.20.0, bitops 1.0-6, caTools 1.17.1, chron 2.3-47, cluster 2.0.4, colorspace 1.2-6, data.table 1.9.6, dichromat 2.0-0, digest 0.6.9, ensembldb 1.4.3, evaluate 0.9, foreign 0.8-66, formatR 1.4, gdata 2.17.0, ggplot2 2.1.0, gridExtra 2.2.1, gtable 0.2.0, gtools 3.5.0, htmltools 0.3.5, httpuv 1.3.3, httr 1.1.0, interactiveDisplayBase 1.10.3, lattice 0.20-33, latticeExtra 0.6-28, magrittr 1.5, matrixStats 0.50.2, mime 0.4, munsell 0.4.3, nnet 7.3-12, plyr 1.8.3, rpart 4.1-10, rtracklayer 1.32.0, scales 0.4.0, shiny 0.13.2, splines 3.3.0, stringi 1.1.1, stringr 1.0.0, survival 2.39-4, tools 3.3.0, truncnorm 1.0-7, xtable 1.8-2, zlibbioc 1.18.0

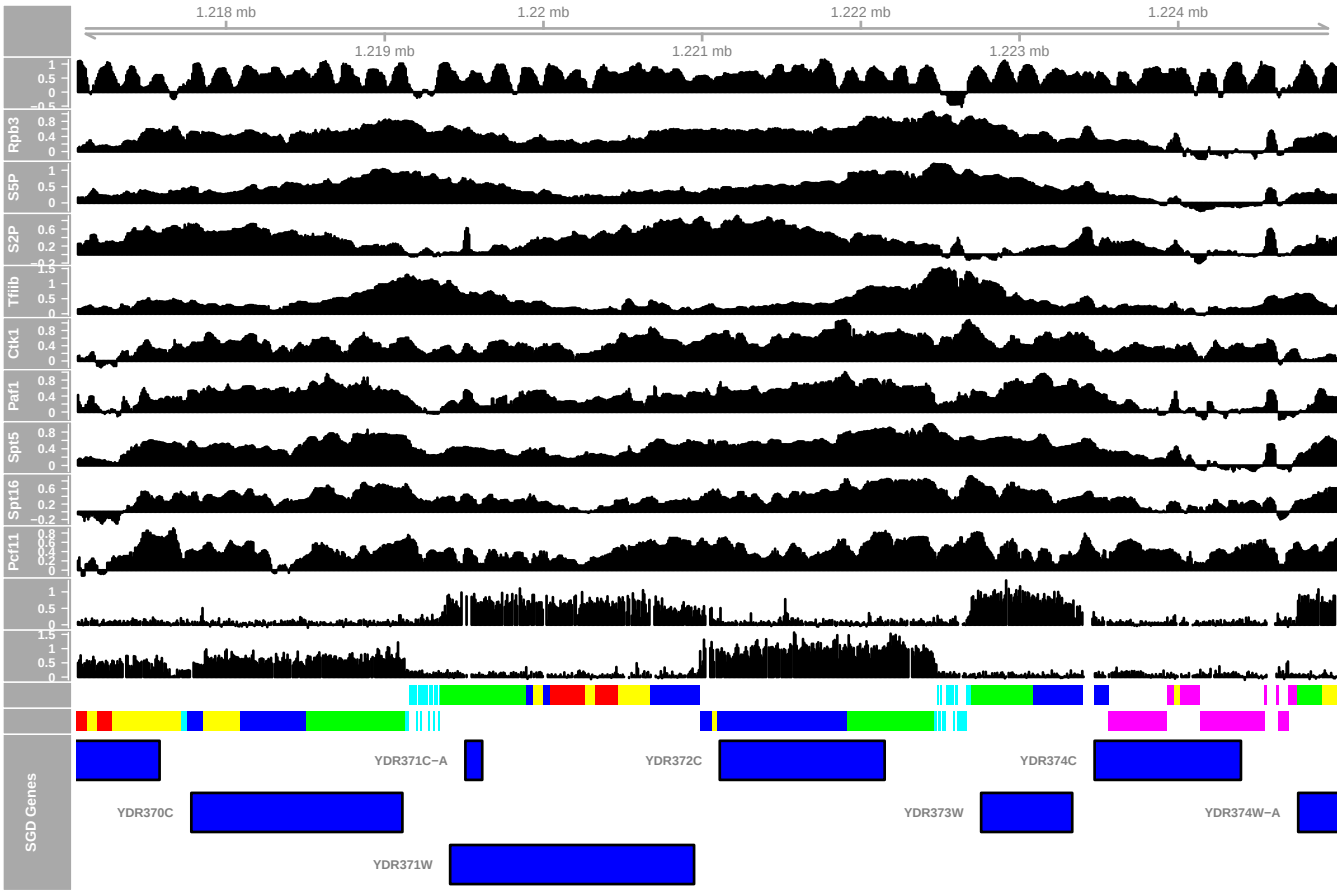


Figure 5: Genome Browser showing the 12 data tracks used for model learning together with the segmentations and known SGD gene annotations.