Chromatin-state discovery and genome annotation with ChromHMM

Jason Ernst[1–5] & Manolis Kellis[6,7]

[1]Department of Biological Chemistry, University of California, Los Angeles, Los Angeles, California, USA. [2]Department of Computer Science, University of California, Los Angeles, Los Angeles, California, USA. [3]Eli and Edythe Broad Center of Regenerative Medicine and Stem Cell Research at University of California, Los Angeles, Los Angeles, California, USA. [4]Jonsson Comprehensive Cancer Center, University of California, Los Angeles, Los Angeles, California, USA. [5]Molecular Biology Institute, University of California, Los Angeles, Los Angeles, California, USA. [6]Broad Institute of MIT and Harvard, Cambridge, Massachusetts, USA. [7]MIT Computer Science and Artificial Intelligence Laboratory, Cambridge, Massachusetts, USA. Correspondence should be addressed to J.E. (jason.ernst@ucla.edu) or M.K. (manoli@mit.edu).

Published online 9 November 2017; doi:10.1038/nprot.2017.124

Noncoding DNA regions have central roles in human biology, evolution, and disease. ChromHMM helps to annotate the noncoding genome using epigenomic information across one or multiple cell types. It combines multiple genome-wide epigenomic maps, and uses combinatorial and spatial mark patterns to infer a complete annotation for each cell type. ChromHMM learns chromatin-state signatures using a multivariate hidden Markov model (HMM) that explicitly models the combinatorial presence or absence of each mark. ChromHMM uses these signatures to generate a genome-wide annotation for each cell type by calculating the most probable state for each genomic segment. ChromHMM provides an automated enrichment analysis of the resulting annotations to facilitate the functional interpretations of each chromatin state. ChromHMM is distinguished by its modeling emphasis on combinations of marks, its tight integration with downstream functional enrichment analyses, its speed, and its ease of use. Chromatin states are learned, annotations are produced, and enrichments are computed within 1 d.

INTRODUCTION

Mapping of epigenomic marks, such as histone modifications, histone variants, regions of open chromatin, and related marks, has emerged as a powerful means to annotate genomes, to identify putative regulatory elements, and to study their changing activity across different cell types and in human disease[1–4]. Individual marks can be studied in isolation, either through aggregation of their genome-wide signal tracks relative to a set of predetermined annotations[5], such as transcription start sites or exon boundaries, or by discovery of narrow peaks or broader domains in which the mark is present in greater frequency than that of the surrounding regions[6]. However, additional information can be gained by studying combinations of multiple marks in their spatial context. Such patterns, termed 'chromatin states', often capture known classes of genomic elements, such as promoters, enhancers, and transcribed, repressed, and repetitive regions[7], and can also capture novel classes or subclasses of elements. Recognizing chromatin states, and identifying their genomic occurrences in each cell type, provides a systematic annotation of DNA elements and regulatory control regions across cell types, which can then be used to interpret genome-wide association studies (GWAS) results, study gene regulation, and analyze cellular differentiation, among many other applications (**Fig. 1**).

Development of the protocol

To address the challenge of interpreting many genome-wide maps of diverse epigenomic marks, we developed a generative machine-learning model to infer 'hidden states' in a given cell type on the basis of the observed epigenomic marks at each genomic position[7]. We termed these hidden states of the genome 'chromatin states', a term that captures both the probabilistic nature of a multistate model and the biological nature of the state of chromatin at that location. We used a multivariate HMM, which allowed the probabilistic modeling of both the combinatorial presence/absence of multiple marks (in the emission parameters of the multivariate HMM) and the spatial constraints of how these mark combinations occur relative to each other across the genome (in the transition matrix of the HMM).

The mark emission probability vector of each state represents the probability with which each mark is found in that state. Dependencies between marks are captured by the different chromatin states, but within each state, mark emission probabilities are assumed to be independent. The transition probabilities from each state to each other state enable the model to capture the positional biases of chromatin states relative to each other, such as broad transcribed or repressed domains. The model parameters are learned from the data *de novo* on the basis of an unsupervised machine-learning procedure that iteratively attempts to maximize the model fit to the data.

We implemented our method in a robust open-source software package, (ChromHMM[8]), which enables the learning of chromatin states, annotates their occurrences across the genome, and facilitates biological interpretation of these states by automatically computing state enrichments for external annotations. Given that histone modification marks occur at the resolution of individual nucleosomes, ChromHMM by default partitions the genome at 200-nucleotide intervals, which roughly correspond to the resolution of a nucleosome and spacer region, although the interval size can be altered by a user-specified parameter. For each genomic interval, ChromHMM then determines the presence or absence of each mark on the basis of the significance of the observed count of sequencing reads relative to a Poisson background distribution, or alternatively accepts user-specified binarization such as those of peak callers. ChromHMM uses the resulting presence–absence calls to learn a chromatin-state model, and create an annotation of state occurrences across the genome. The ChromHMM software includes parallelization support for multiprocessor machines, which can substantially reduce run-time.

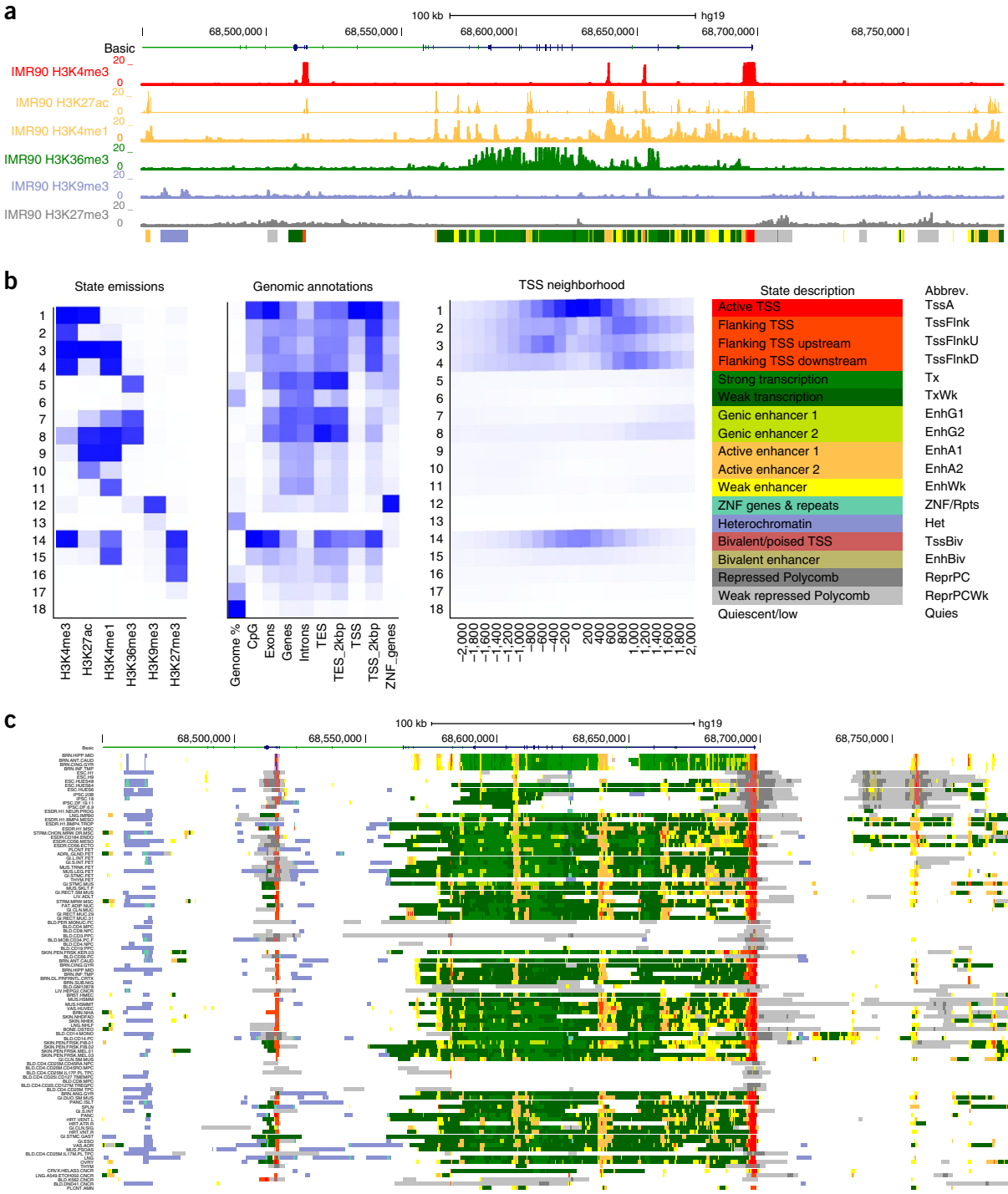

Figure 1 | Overview of ChromHMM. (**a**) Tracks of multiple histone modifications are shown from one cell type, IMR90. From such types of tracks, ChromHMM learns a set of chromatin-state definitions *de novo*, and then assigns each location in the genome to an instance of each state. The chromatin-state assignments for IMR90 made on the basis of the model in **b**) are shown below the histone modifications. (**b**) The panel on the left displays a heatmap of the emission parameters in which each row corresponds to a different state, and each column corresponds to a different mark for the Roadmap Epigenomics 18-state expanded model defined on the basis of the observed data for six histone modifications (H3K4me1, H3K4me3, H3K27me3, H3K27ac, H3K36me3, and H3K9me3) from ref. 3. The darker blue color corresponds to a greater probability of observing the mark in the state. The heatmap to the right of the emission parameters displays the overlap fold enrichment for various external genomic annotations in IMR90 cells (epigenome E017) and is similar to what was previously shown for H1-hESC cells in ref. 3. A darker blue color corresponds to a greater fold enrichment for a column-specific coloring scale. The heatmap to the right of that shows the fold enrichment for each state for each 200-bp bin position within 2 kb around a set of transcription start sites (TSSs). A darker blue color corresponds to a greater fold enrichment, and there is one color scale for the entire heatmap. Candidate-state descriptions for each state, followed by a state abbreviation, are shown to the right of the heatmaps. (**c**) The panel displays the browser view of ChromHMM genome annotation on the basis of the model in **b**, which was defined across 98 cell and tissue types[3]. Each row corresponds to one of the cell or tissue types.

Applications of the method

We originally applied the chromatin-state modeling framework to genome-wide ChIP-seq maps of 41 epigenomic modifications and related factors from a single-cell type, CD4T cells[5,7,9], showing that their combinations and relative genomic positions were highly informative of distinct biological functions. We later demonstrated the applicability of the method to simultaneous annotation of multiple cell types by virtual concatenation of nine genome-wide epigenomic maps, each from nine cell types, allowing us to learn a common set of chromatin states and their locations across the nine cell types[1] (**Fig. 2**). We subsequently scaled this approach to >100 cell and tissue types (**Fig. 1b**,**c**)[3], and we also showed that open-chromatin assays can also be used as features[10,11]. When not all marks are available in all cell types, ChromHMM can make the best assignments with the available data. Alternatively, ChromHMM can also be applied to imputed data produced by a program such as ChromImpute[12], which enables chromatin-state annotations to be produced on the basis of a consistent set of marks across a concatenated set of cell types, even if each mark is not mapped in every cell type considered. Another approach for applying ChromHMM to multiple cell-type data is to stack the tracks from the multiple cell types over the same genomic location[8,13,14] (**Fig. 2**).

ChromHMM has also been widely used by many others in diverse applications, including studies of the noncoding genome, gene regulation, and human disease. The ChromHMM software has been downloaded thousands of times and has been cited in the literature more than 500 times as of April 2017. Large consortium projects, including ENCODE[10], Roadmap Epigenomics[3], Blueprint[15], CEEHRC[16], Mouse ENCODE[17], and Drosophila modENCODE[18], have used ChromHMM as a basis for their analyses. ChromHMM reference annotations for humans are incorporated as reference tracks in popular genome browsers, including the UCSC Genome Browser[19] and Ensembl[20]. In the context of gene regulation, ChromHMM has been used for understanding long-range chromatin interactions[21], nascent transcripts[22], cellular reprogramming[14,23], topologically associated domains[24], transcription-factor binding preferences[1,25], and regulatory motif disruptions in massively parallel reporter assays[26]. In the context of human disease, ChromHMM annotations have been analyzed in conjunction with GWAS for predicting the cell types relevant to human traits or predicting the mechanism of individual GWAS-identified loci[1,4,10,27]. ChromHMM has had applications to studying diseases such as Alzheimer's disease[28,29], neurodegeneration[30], type 2 diabetes[31], and cancer[32–35], among many other diseases. ChromHMM has also been used for studying chromatin-state variation across individuals[36], chromatin-state changes across different *Drosophila* species[37], and aging[38,39], among many other applications.

In this protocol, we will focus on running ChromHMM to obtain a new chromatin-state model, and constructing genome-wide annotations, rather than the diverse applications that the resulting annotations enable. We also note that for many of these applications, a researcher can simply use existing chromatin-state annotations available for >100 cell or tissue types on the basis of the data generated by the Roadmap Epigenomics[3], ENCODE[10], and Blueprint[15] projects (**Table 1**). These chromatin-state annotations can be accessed through multiple different browsers and web portals (**Table 1**). In addition, chromatin-state annotations for single-nucleotide polymorphisms (SNPs) can be queried directly through databases, including HaploReg[40] and RegulomeDB[41].

Comparison with other methods

ChromHMM is among the first, most widely used, and most widely cited tools for chromatin-state discovery and annotation, but a number of other software packages are now available that address this problem. Before ChromHMM, HMMSeg[42] used a HMM to partition the genome into two states[43]. Segway was later developed by the same group, using dynamic Bayesian networks, and was used to generate annotations on the basis of

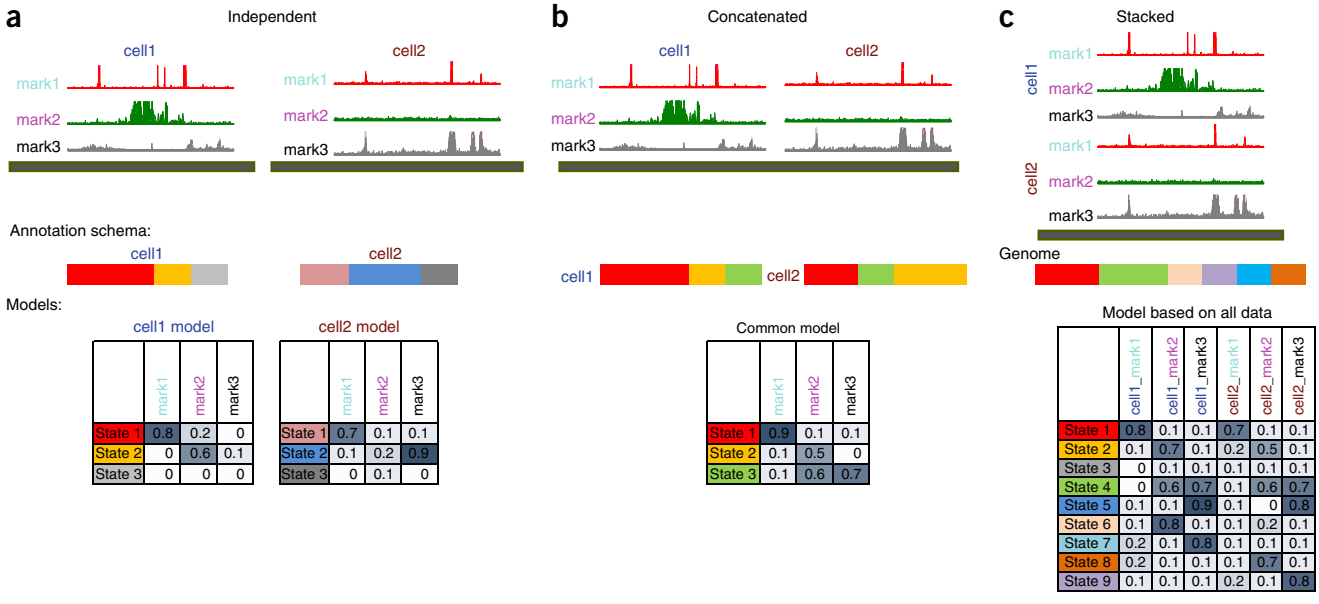

Figure 2 | Overview of different options for handling multiple cell types in ChromHMM. (**a**) Multiple cell types are treated independently, leading to a different model and annotation for each cell type. (**b**) Multiple cell types are effectively concatenated, leading to one shared model for all cell types, but cell-type-specific annotations. (**c**) Data from multiple cell types are stacked, leading to one model based on an expanded set of features, and one annotation of the genome.

TABLE 1 | Summary of some major available ChromHMM reference models and annotations defined across multiple cell and tissue types.

Model	Input marks	Cell/Tissue types	Assembly	No. of states	Ref.	URLs
Roadmap Epigenomics core model	5-marks-observed human data (H3K4me1, H3K4me3, H3K27me3, H3K9me3, and H3K36me3)	127 diverse ones (111 Roadmap Epigenomics, 16 ENCODE)	hg19	15	3	http://compbio.mit.edu/roadmap https://www.encodeproject.org http://epigenomegateway.wustl.edu/ http://genome.ucsc.edu/cgi-bin/hgTracks?db=hg19&hubUrl=http://vizhub.wustl.edu/VizHub/RoadmapIntegrative.txt
Roadmap epigenomics expanded model	6-marks-observed human data (H3K4me1, H3K4me3, H3K27me3, H3K9me3, H3K36me3, and H3K27ac)	98 diverse ones (82 Roadmap Epigenomics, 16 ENCODE)	hg19	18	3	http://compbio.mit.edu/roadmap https://www.encodeproject.org http://epigenomegateway.wustl.edu/ http://genome.ucsc.edu/cgi-bin/hgTracks?db=hg19&hubUrl=http://vizhub.wustl.edu/VizHub/RoadmapIntegrative.txt
Roadmap epigenomics imputation-based model	12-marks-imputed human data (H3K4me1, H3K4me3, H3K27me3, H3K9me3, H3K36me3, H3K27ac, H3K9ac, H4K20me1, H3K79me2, H3K4me2, H2A.Z, and UW DNaseI)	127 diverse ones (111 Roadmap Epigenomics,16 ENCODE)	hg19	25	12	http://compbio.mit.edu/roadmap http://epigenomegateway.wustl.edu/ http://genome.ucsc.edu/cgi-bin/hgTracks?db=hg19&hubUrl=http://vizhub.wustl.edu/VizHub/RoadmapIntegrative.txt
ENCODE integrative analysis ChromHMM model	14-marks-observed human data (H3K4me1, H3K4me3, H3K27me3, H3K36me3, H3K27ac, H3K9ac, H4K20me1, H3K4me2, Pol2, CTCF, FAIRE-seq, UW DNaseI, Duke DNaseI, and Input)	6 ENCODE cell types	hg19	25	11,25	https://genome.ucsc.edu/cgi-bin/hgTrackUi?g=wgEncodeAwgSegmentation&db=hg19
Ernst *et al.*, 2011 *Nature* model	10-marks-observed human data (H3K4me1, H3K4me3, H3K27me3, H3K36me3, H3K27ac, H3K9ac, H4K20me1, H3K4me2, CTCF, and Input)	9 ENCODE cell types	hg18 and hg19 liftover	15	1	https://genome.ucsc.edu/cgi-bin/hgTrackUi?g=wgEncodeBroadHmm&db=hg19 https://www.encodeproject.org

(continued)

Ensembl human models	Minimum of 6-marks-observed human data (H3K4me1, H3K4me3, H3K27me3, H3K9me3, H3K36me3, and H3K27ac)	74 Cell types from Blueprint, Roadmap Epigenomics, and ENCODE	hg38	25	15,20	http://ensembl.org/info/genome/funcgen/regulatory_segmentation.html http://ftp.ensembl.org/pub/release-87/data_files/homo_sapiens/GRCh38/segmentation_file/085/
Mouse ENCODE	4-marks-observed mouse and human data (H3K4me1, H3K4me3, H3K36me3, and H3K27me3)	15 mouse ENCODE cell types and 6-ENCODE cell types	mm9 and hg19	7	17	https://www.encodeproject.org http://main.genome-browser.bx.psu.edu/cgi-bin/hgTrackUi?g=meryChromHmm7s&db=mm9

more states[44]. A number of other tools have appeared since for learning chromatin states and annotating the genome, including TreeHMM[45], GATE[46], diHMM[47], CMINT[48], hiHMM[49], IDEAS[50], Segway-GBR[51], STAN[52], GenoSTAN[53], EpiCSeg[54], and Spectacle[55], some of which are tailored to specific use cases, such as strand-specific annotations[52], hierarchical modeling[47], or reduced running time, but with a decreased ability to distinguish different types of broader domains[55]. Additional chromatin-state methods and models have been proposed, but these were not accompanied by new publicly available software applications designed for the task[56–63]. Additional tools exist that aim at addressing related problems in unsupervised epigenome annotation: jMOSAiCS[64]

Box 1 | Summary of main ChromHMM commands

This box summarizes some of the main commands of ChromHMM. Additional details about these commands and other commands can be found in the ChromHMM manual. Additional details about some of the output formats are presented in **Box 6**. For the file naming, '`N`' corresponds to the number of states and '`celltype`' corresponds to one of the cell types.

`BinarizeBam`—This command converts a set of BAM files of aligned reads into binarized data files in a specified output directory, which can then be used as input to `LearnModel`.

`BinarizeBed`—This command is similar to `BinarizeBam`, but takes aligned reads in BED format instead of BAM format.

`BinarizeSignal`—This command takes a set of signal files, by default assumed to represent counts, and converts them into binarized data in a specified output directory, which can then be used as input to `LearnModel`.

`LearnModel`—This command takes as input a set of binarized data files and, when executed, allows ChromHMM to learn a chromatin-state model. After learning the model, ChromHMM produces a genome annotation in both a simple four-column BED format and BED formats designed for view in a genome browser. After producing the genome annotation, ChromHMM also computes overlap and neighborhood enrichments for a set of external annotations. In addition to prompting ChromHMM to learn the model, this command also effectively executes the `MakeSegmentation`, `MakeBrowserFiles`, `OverlapEnrichment`, and `NeighborhoodEnrichments` commands. A webpage is created with links to all the files and images created (see **Box 6**, and **Fig. 3**).

`MakeSegmentation`—This command takes a previously learned model and binarized data, and outputs for each cell type a segmentation file containing a genome annotation in a four-column BED format in files named `celltype_N_segments.bed`.

`MakeBrowserFiles`—This command takes as input a segmentation file containing a genome annotation in a four-column BED format and converts it into two types of BED formats for viewing in the browser, one in which a genome annotation can be displayed on a single line and the other in which there is one line per state, named `celltype_N_dense.bed` and `celltype_N_expanded.bed`, respectively, when called by `LearnModel`.

`OverlapEnrichment`—This command computes the fold enrichment of each state of a ChromHMM-generated annotation for a set of external annotations, generating the output in text and image formats, and producing files named `celltype_N_overlap.txt`, `celltype_N_overlap.png`, and `celltype_N_overlap.svg` when called by `LearnModel`.

`NeighborhoodEnrichment`—This command computes the fold enrichment of each state relative to a set of anchor positions, generating the output in text and image formats, and producing files named `celltype_N_anchorfeature_neighborhood.txt`, `celltype_N_anchorfeature_neighborhood.png`, and `celltype_N_anchorfeature_neighborhood.svg` when called by `LearnModel`.

`CompareModels`—This command compares the emission parameters of models with different numbers of states to a reference model in terms of parameter correlations. The reference model is usually selected to be the model with the largest number of states being considered. The output contains, for each state of the reference model and for each other model being compared, the maximum correlation of the state of the reference model with any state of the model being compared. The output is generated in .txt, .png, and .svg image formats.

`Reorder`—This command reorders states of the model, or the columns of the emission matrix, and outputs updated model parameter files.

© 2017 Macmillan Publishers Limited, part of Springer Nature. All rights reserved.

and scHMM[65] call peaks or domains for epigenomic marks in the context of other marks, but annotate only peaks or explicitly observed combinations of them, which can grow exponentially with the number of marks; ChromaSig[66] discovers multi-mark epigenomic patterns in fixed-size windows, but leaves large portions of the genome unannotated to any pattern.

ChromHMM has several characteristics that distinguish it from most methods in this vibrant space: (i) Although most other methods seek to model the signal levels of each mark, which can lead to overfitting small signal variation due to noise, ChromHMM focuses its modeling power on combinations of epigenomic marks, by using binary presence/absence input features. This has enabled ChromHMM to discover chromatin states such as a state associated with Zinc finger genes (despite the diffuse signal of its associated marks and small genome coverage)[3,7,12] and a putative bivalent promoter state[67] that are often missed by other methods that directly model signal levels, even when applied on the same data sets[53]. (ii) ChromHMM's robust and efficient implementation, including multi-core parallelization, enables it to be used for large-scale applications, as demonstrated by learning models based on a dozen marks across >100 cell and tissue types while using the entire genome for training[12]. (iii) ChromHMM can work directly from aligned reads, so that separate signal-generation software is not needed; and (iv) ChromHMM also has practical advantages, including its ease of use, easy installation (it requires only unzipping a .zip file), and broad portability (it requires only Java, which is commonly installed on many systems, and has no external dependencies or additional system requirements).

ChromHMM also provides tight integration with downstream chromatin-state enrichment analyses, including sets of external annotation files for commonly studied species, which facilitates biological interpretation of the discovered chromatin states. After producing a chromatin-state annotation, ChromHMM also automatically provides a report that includes the model parameters, state enrichments for external annotations, and chromatin-state annotations in formats for both downstream computational analysis and browser visualization.

Level of expertise needed to implement the protocol

ChromHMM uses a command-line interface, which facilitates scripting but requires a minimum level of familiarity with command-line execution of programs. It also generates reports of model parameters and enrichments for external genomic annotations, which requires some familiarity with genome biology to help in the biological interpretation of the results. Beyond basic scripting and some understanding of genomics, no other specialized skills are needed.

Limitations of ChromHMM

ChromHMM can automatically infer models and compute enrichments for the states of the models on the basis of external annotations without prior biological knowledge; however, the state interpretation still requires a knowledgeable human. The interpretations that one gives to states are only candidate-state annotations, and one should not assume that each instance assigned to a state is a true instance of the annotation.

Although ChromHMM has a `CompareModels` command that facilitates comparison of emission parameters of models with a different number of states, it does not automatically decide on the number of states one should use. Although sometimes there can be a clear lower bound on the number of states, as key biological-state distinctions are not resolved with fewer states, the upper bound can be less clear. In general, the number of biologically meaningful chromatin states that can be discovered will increase with the number of input tracks, although this will depend on the extent of redundant information in the tracks. Finally, even for a fixed set of input tracks, the desired resolution of biological interpretation can dictate the number of states.

ChromHMM works on fixed bin sizes (default 200 bp), but for some applications, one can obtain more relevant higher-resolution boundaries by directly considering nucleosome depletion from the histone modification signal[1], DNase I peaks[68], or footprints within them[69,70]. These boundaries could then still be used in conjunction with chromatin-state annotations[71].

Experimental design

ChromHMM commands. ChromHMM has a number of different high-level commands (**Box 1**) that provide support for binarizing aligned reads, learning models based on the binarized data, and conducting downstream analyses.

Input data. ChromHMM is flexible in the types of data that it can model. Any type of data that can be binarized in a meaningful way can, in principle, be applied to ChromHMM. ChromHMM's provided binarization procedure uses a given set of aligned reads. If desired, manipulations to reads, such as collapsing duplicate reads or downsampling an imbalanced and deeply sequenced data set, should be conducted before providing the reads to ChromHMM.

ChromHMM can use input control data, such as those produced by mapping whole-cell extracts or IgG ChIP data[72]. Control data provide information on local background signal levels, which ChromHMM can use to adjust binarization thresholds locally instead of having a uniform threshold genome wide. ChromHMM can also use control data as an additional model feature.

MATERIALS

EQUIPMENT

Software

- ChromHMM (mirrored at http://www.biolchem.ucla.edu/labs/ernst/ChromHMM and http://compbio.mit.edu/ChromHMM) can be run on any system supporting Java 1.5 or a later version (https://java.com/en/download/). A user manual with more detailed documentation is also available from the ChromHMM website. The software, including the source code and documentation, is also available through a version-control repository at https://github.com/jernst98/ChromHMM. This protocol is based on the most recent release of ChromHMM (v1.12, release date: April 15, 2016)

Input data

- The recommended input data to ChromHMM are coordinates of aligned reads in BAM, BED, or tagAlign formats. If aligned reads are unavailable, such as with imputed data, ChromHMM also provides support for externally called peaks or signal data. If you are using externally called peaks, then they should be narrow peak calls, as the use of broad peak calls is not recommended, as described in Step 4. If you are using signal data, ChromHMM provides support for binarizing it through the `BinarizeSignal` command, which by default assumes that the signal represents count data. The use of `BinarizeSignal` is discussed in the user manual. For all text files provided as input, ChromHMM supports both unzipped files and gzipped files with a .gz extension.

© 2017 Macmillan Publishers Limited, part of Springer Nature. All rights reserved.

PROCEDURE

Installation ● TIMING 5 min

1| If Java is not already installed, then install Java from https://java.com/en/download/ and follow its corresponding instructions.

2| Download ChromHMM from http://www.biolchem.ucla.edu/labs/ernst/ChromHMM or http://compbio.mit.edu/ChromHMM/. This will save a file (ChromHMM.zip) to a local computer. After the file has been saved, unzip the file.

3| From the command line, change into the now-unzipped ChromHMM subdirectory containing the ChromHMM.jar file.

Binarization ● TIMING 1 h

4| Prepare a tab-delimited text file specifying the design file for the cell-mark-aligned read files according to whether there is a single cell type (option A) or multiple cell types that will be treated independently (option B) or concatenated (option C), or multiple cell types for which the features will be stacked to provide a single-genome annotation (option D) (**Fig. 2**).

(A) Single-cell type

(i) Format a single file in the form:

```
cell1    mark1    cell1_mark1_file
cell1    mark2    cell1_mark2_file
```

(B) Multiple cell types being treated independently

(i) Format a file for each cell type in the form:

File 1:

```
cell1    mark1    cell1_mark1_file
cell1    mark2    cell1_mark2_file
```

File 2:

```
cell2    mark1    cell2_mark1_file
cell2    mark2    cell2_mark2_file
```

(C) Multiple cell types being concatenated

(i) Format a single file in the form:

```
cell1    mark1    cell1_mark1_file
cell1    mark2    cell1_mark2_file
cell2    mark1    cell2_mark1_file
cell2    mark2    cell2_mark2_file
```

If some mark is specified only in a subset of the cell types, then it will be treated as missing in the remaining cell types (encoded with '2') in the binarized files.

(D) Multiple cell types with stacking features

(i) Format a single file in the form:

```
genome    cell1_mark1    cell1_mark1_file
genome    cell1_mark2    cell1_mark2_file
genome    cell2_mark1    cell2_mark1_file
genome    cell2_mark2    cell2_mark2_file
```

▲ **CRITICAL STEP** If there are multiple files for the same cell and mark combination, then each can be specified and their reads will be pooled, or alternatively, the files can be combined outside of ChromHMM.

▲ **CRITICAL STEP** The .bed or .bam files supplied to `BinarizeBed` or `BinarizeBam` should in general contain the locations of aligned reads. ChromHMM also allows binarizing data on the basis of already-called peaks, as opposed to aligned reads, in which case the '-peaks' flag would need to be specified when executing the `BinarizeBam` or `BinarizeBed` commands in Step 7; otherwise, the output would not be meaningful. If you are supplying already-called peaks to ChromHMM, it is not recommended to use broad peak calls. ChromHMM already considers spatial information, and if you are supplying broad peaks, then ChromHMM could have states corresponding to a combination of marks that do not actually co-occur at the resolution at which it operates.

5| If input control data such as whole-cell extract or IgG ChIP data[72] are available, consider using them either as an additional feature (option A) or by adjusting the binarization threshold locally on the basis of the relative level of the control reads (option B)[8]. Option A is recommended if the control data are relatively flat, except in specific positions (often associated with artifacts), or if the control data were not sequenced deeply. Option B is recommended if the control data show peaks in regions other than artifact-associated regions and are sequenced sufficiently deeply to estimate accurate levels of the local background.

(A) Treat control data as an additional feature

(i) Format the cell-mark-aligned read file(s) to have the form:

```
cell1    mark1      cell1_mark1_file
cell1    mark2      cell1_mark2_file
cell1    control    cell1_control_file
```

(B) Use control data to adjust the binarization threshold locally

(i) Format the cell-mark-aligned read file(s) to have the form:

```
cell1    mark1    cell1_mark1_file    cell1_control_file
cell1    mark2    cell1_mark2_file    cell1_control_file
```

in which the control file is the fourth column. Mark-specific control files can also be specified.

6| Determine if the genome assembly corresponding to the data has a chromosome-length file in the CHROMSIZES directory. Chromosome-length files for the assemblies hg18, hg19, hg38, mm9, mm10, rn5, rn6, danRer7, danRer10, dm3, dm6, ce6, and ce10 are currently provided in the CHROMSIZES directory. If the desired assembly is already included, nothing needs to be done at this step.

If the desired assembly is not found in the directory, then it must be prepared and placed in the CHROMSIZES directory. The format is a two-column format in which the first column is the chromosome name and the second is the chromosome length. Such information for a number of additional assemblies is available from the UCSC Genome Browser[73]. Linux and Mac users can obtain such files directly by first downloading the `fetchChromSizes` script from the corresponding directory for their system here: http://hgdownload.cse.ucsc.edu/admin/exe/. At the command prompt, type

```
./fetchChromSizes assembly > assembly.txt
```

7| Execute the command to binarize the data. If the reads are in BED or in tagAlign format, execute the BinarizeBed command (option A). If the reads are stored in BAM files, execute the command BinarizeBam (option B).

(A) Execute BinarizeBed

(i) Enter the command

```
java -mx4000M -jar ChromHMM.jar BinarizeBed chrlengthfile inputdir cellmarkfiletable outputdir
```

`chrlengthfile` contains the chromosome-length file described in Step 6, `inputdir` is the directory with the aligned reads, `cellmarkfiletable` is the design file described in Steps 4 and 5, and `outputdir` is the directory in which the binarized data should be written. For each cell type and chromosome, ChromHMM will generate one binarized data file. The first line will contain the cell type and chromosome. The second line will specify the marks, and the remaining lines for each consecutive bin will specify the binary call for each mark. '`-mx4000M`' specifies the amount of memory given to Java and should be adjusted as needed for this and other commands.

When executing the command, also add any appropriate optional parameters. Optional parameters are specified immediately after `BinarizeBam` or `BinarizeBed`. The full set of options is discussed in the manual. Some options to note are described in **Box 2**.

? TROUBLESHOOTING

(B) Execute BinarizeBam

(i) Enter the command

```
java -mx4000M -jar ChromHMM.jar BinarizeBam chrlengthfile inputdir cellmarkfileta-
ble outputdir
```

The parameters for `BinarizeBam` are the same as those for `BinarizeBed` mentioned above.

? TROUBLESHOOTING

File preparation for enrichment analyses ● TIMING 30 min

8| If you are interested in including external coordinate annotations not already included with ChromHMM in the overlap enrichment analysis of the ChromHMM genome annotation, then prepare additional external coordinate files for this analysis. ChromHMM includes some files for overlap enrichment analysis for the assemblies listed in Step 6. If you are using one of these assemblies, then add additional files into the corresponding assembly subdirectory within the `COORDS` directory. If you are using a different assembly, first create a subdirectory for that assembly within the `COORDS` directory. The coordinate files should be in .bed format. This step can also be done after executing the `LearnModel` command in Step 10 with enrichments then computed by running `OverlapEnrichment` in Step 13, although the overlap enrichment analysis would not be automatically included in the report generated by the `LearnModel` command.

Box 2 | Notable options for the 'BinarizeBed' and 'BinarizeBam' commands

`-b binsize`—This specifies the bin size that ChromHMM will use. The default is 200-bp bins. If you are setting the size to a different value, then the same option and value must be included with other commands. If you are using the default ChromHMM binarization procedure, this value should not be made too small; otherwise, there will be too few reads to reliably make presence calls.
`-p poissonthreshold`—This specifies the tail probability of the Poisson distribution to which the binarization threshold should correspond. The default value is 0.0001.
`-f foldthresh`—This parameter specifies the minimum fold enrichment threshold for the signal to have a present call, even if the Poisson significance threshold is met. By default, this value is 0, but setting it to larger values may be useful when working with deeply and imbalanced sequenced data sets.
`-n shift`—This parameter specifies the amount reads are shifted to determine their bin assignment. Reads are shifted in a strand-aware way from the 5′ end in the direction of the 3′ end. The default read shift is 100 bp. ChromHMM directly supports only a single read shift value for all marks. If different shift amounts are needed for different marks, they should be binarized separately and then merged together outside of ChromHMM.
`-center`—Specify this flag if reads should be placed in the bin on the basis of the center coordinate position.
`-peaks`—As discussed in Step 4, include this flag if the .bed or .bam files correspond to peak calls instead of aligned reads. It is not recommended to use peak calls based on broad domains as input.

9| If you are interested in including external position annotations not already included with ChromHMM in the neighborhood enrichment analysis of the ChromHMM genome annotation, then prepare additional external position files for this analysis. ChromHMM includes some files for the assemblies listed in Step 6. If you are using one of these assemblies, then add additional files into the corresponding assembly subdirectory within the `ANCHORFILES` directory. If you are using a different assembly, first create a subdirectory for that assembly within the `ANCHORFILES` directory. The format of the file should be a tab-delimited text file in which the first column is the chromosome, the second column is the coordinate, and optionally, the third column is the strand ('+' or '–'). This step can also be done after executing `LearnModel` in Step 10 with enrichments then computed by running `NeighborhoodEnrichment` in Step 14, although the neighborhood enrichment analysis would not be automatically included in the report generated by the `LearnModel` command.

Model learning ● TIMING typically ~4 h, but can be highly variable

10| To start model learning and have automatic enrichments computed after the model is learned, execute the `LearnModel` command:

```
java -mx4000M -jar ChromHMM.jar LearnModel inputdir outputdir numstates assembly
```

In this command, `inputdir` specifies the directory containing the binarized data, which will often be the output directory from Step 7. `outputdir` is the directory in which the output files from the `LearnModel` command should go. `numstates` specifies the number states. `assembly` specifies the genome assembly for the input.

When executing the commands, also add any appropriate optional parameters. Optional parameters are specified immediately after `LearnModel`. The full set of options and additional details are discussed in the manual. Some options to note are described in **Box 3**.

? TROUBLESHOOTING

11| Repeat Step 10 for models with different numbers of states. If a computer cluster is available, then model learning can proceed in parallel.

Post-model-learning analysis ● TIMING 2 h

▲ CRITICAL The steps in this section are optional; users can choose any or all steps depending on how they wish to use and analyze the learned models.

12| If you are interested in comparing models with different numbers of states at the emission parameter level, which can complement comparisons at the enrichment level, use the `CompareModels` command to generate a heatmap comparison by entering the following command:

```
java -mx4000M -jar ChromHMM.jar CompareModels referencemodel comparedir outputprefix
```


© 2017 Macmillan Publishers Limited, part of Springer Nature. All rights reserved.

Box 3 | Notable options for the 'LearnModel' command

`-b binsize`—This specifies the bin size and it should match what is specified for `BinarizeBam` or `BinarizeBed`.
`-p maxprocessors`—This parameter specifies the maximum number of processors that ChromHMM should use. To have ChromHMM try to use as many processors as those to which it has access, add the `-p 0` flag. If the `-p` option is specified, ChromHMM uses a parallel model-learning procedure, which can substantially reduce run-time when multiple processors are available. If the `-p` option is not specified, then ChromHMM uses a single processor and an alternative model-learning procedure designed for a single processor.
`-init information | random | load`—This parameter specifies the approach to initializing the parameters of the model. `information` is the default approach[8]. `random` initializes the parameter randomly on the basis of a random seed. `load` initializes the parameters on the basis of the contents of a model file, with smoothing away from 0 by default. Note that for the `information` initialization, the number of states cannot be greater than the number of observed mark combinations.
`-r maxiterations`—This parameter specifies the maximum number of iterations over all the input in the model learning. The default is 200.
`-printstatebyline`—This flag directs ChromHMM to also output the state assignment of each bin printed as one line per bin in consecutive order. The first two lines are header lines. These files end with `_maxstate.txt`.
`-printposterior`—This flag directs ChromHMM to also output the posterior probabilities over state assignments for each bin. The first two lines are header lines, and then each consecutive line contains the posterior probability for each state in order in tab-delimited format. These files end with `_posterior.txt`.

The `referencemodel` parameter specifies a file that contains the emission parameters of a reference model to which other models should be compared. Generally, this will be a model with the largest number of states being considered. The file should start with `'emissions_'` and end with `'.txt'` or `'.txt.gz'`. The `comparedir` parameter specifies the directory that has the model emission parameters that should be compared with the `referencemodel`. The files should start with `'emissions_'` and end with `'.txt'` or `'.txt.gz'`. `outputprefix` contains the prefix of the output files, including possibly a directory. `'.txt'`, `'.svg'`, and `'.png'` are appended to this prefix to produce files containing the output (**Box 1**).

13| If you are interested in computing overlap enrichments in addition to those that were already computed after applying the `LearnModel` command, then use the `OverlapEnrichment` command by entering

```
java -mx4000M -jar ChromHMM.jar OverlapEnrichment inputsegment inputcoorddir
outfileprefix
```

In this command, `inputsegment` specifies the segmentation file for which overlap enrichments should be computed. `inputcoorddir` specifies the directory containing the external coordinates for overlap enrichment analysis. These files should be prepared according to Step 8. `outfileprefix` contains the prefix, including possibly the directory, of the output files that will have the extensions `'.txt'`, `'.png'`, and `'.svg'` added to them (**Box 1**).

When executing the commands, also add any appropriate optional parameters. Optional parameters are specified immediately after `OverlapEnrichment`. The full set of options is discussed in the manual. Some options to note are described in **Box 4**.

Box 4 | Notable options for the 'OverlapEnrichment' command

`-b binsize`—This specifies the bin size and it should match what was used in the `LearnModel` command.
`-center`—This directs ChromHMM to compute the enrichment on the basis of the center position only of each interval in the external coordinate file. Using this option, each entry has effectively equal weight, whereas, by default, entries that are wider will be given greater weight.
`-multicount`—This specifies that bases overlapped by multiple intervals should be counted multiple times. The default is to count a base only once.
`-m labelmappingfile`—This specifies a tab-delimited column mapping state IDs to descriptive names to display them on the heatmap. The first column contains the state ID, including the prefix, and the second column contains the descriptive name.
`-uniformscale`—This specifies that the color scale should be uniform for the entire heatmap. The default is to use a column-specific coloring scale.

© 2017 Macmillan Publishers Limited, part of Springer Nature. All rights reserved.

Box 5 | Notable options for the 'NeighborhoodEnrichment' command

`-b binsize`—This specifies the bin size and it should match what was used in the `LearnModel` command.
`-l numleftintervals`—This specifies the number of columns to the left of the anchor positions for which enrichments should be shown. The default is 10.
`-r numrightintervals`—This specifies the number of columns to the right of the anchor positions for which enrichments should be shown. The default is 10.
`-s spacing`—This specifies the spacing in base pairs at which enrichments should be displayed. The default is 200.

14| If you are interested in computing neighborhood enrichments other than those that were already computed after applying the `LearnModel` command, then use the `NeighborhoodEnrichment` command by entering

```
java -mx4000M -jar ChromHMM.jar NeighborhoodEnrichment inputsegment anchorpositions outfileprefix
```

Box 6 | Default output files for the ChromHMM LearnModel command

For the following files, `N` indicates the number of states, and `celltype` indicates the cell type. If you are applying ChromHMM in concatenated mode to multiple cell types, there will be corresponding files for each cell type.
`webpage_N.html`—This is a webpage that contains the commands used to learn the model and links to all the files generated as part of the output.
`emissions_N.txt`—This is a tab-delimited text file containing the emission parameters of the HMM. The first row is a header row indicating each mark used to define the model, and the first column indicates the state. The values correspond to the probability of observing the mark of the column, conditioned on being in the state of the row.
`emissions_N.png`,`emissions_N.svg`—These files display the contents of `emissions_N.txt` as a heatmap in .png and .svg image formats, respectively.
`transitions_N.txt`—This is a tab-delimited text file containing the transition parameters of the model. The first row and first column are header rows and columns, respectively. A value corresponds to a transition probability from the state of the row to the state of the column.
`transitions_N.png`,`transitions_N.svg`—These files display the contents of `transitions_N.txt` as a heatmap in .png and .svg image formats, respectively.
`model_N.txt`—This file contains the emission, transition, and initial parameters of the HMM in a format for which ChromHMM can parse all the parameters.
`celltype_N_segments.bed`—This file is a segmentation of the genome containing ChromHMM's genome annotation for a cell type in an easy-to-parse format. This is a four-column file in .bed format. The first column gives the chromosome, the second column gives the beginning coordinate of the segment, the third column gives the end coordinate of the segment, and the fourth column gives the state. The fourth column includes both the state prefix, which indicates how the states were ordered, and the state number. The state prefixes are 'E' for emission-based ordering, 'T' for transition-based ordering, and 'U' for user-specified ordering.
`celltype_N_dense.bed`—This file gives ChromHMM's genome annotation in a format that can be loaded into a browser and displayed on a single line, with different states being differentiated by different colors.
`celltype_N_expanded.bed`—This file gives ChromHMM's genome annotation in a format that can be loaded into a browser and displayed such that each state appears on a different line.
`celltype_N_overlap.txt`—This file gives the enrichment of the states for various external annotations. The first row and the first column are header rows and columns, respectively. The second column contains the percentage of the genome each state covers. The last row indicates the percentage of the genome that each external annotation covers. The remaining rows correspond to different states, and the remaining columns correspond to different external annotations. The values in these rows and columns correspond to the fold enrichments for the presence of the external annotation in the state. The fold enrichments are computed as the ratio of the fraction of bases assigned to the state that are in the external category to the fraction of bases in the genome that are in the external category.
`celltype_N_overlap.png`,`celltype_N_overlap.svg`—These files display the contents of `celltype_N_overlap.txt` in .png and .svg image formats, respectively, except without the bottom row containing the base percentages.
`celltype_N_anchorfeature_neighborhood.txt`—This file contains the fold enrichment of each chromatin state at fixed positions relative to a set of 'anchorfeature' positions, in which each anchorfeature corresponds to one coordinate file provided to the neighborhood enrichment analysis.
`celltype_N_anchorfeature_neighborhood.png`,`celltype_N_anchorfeature_neighborhood.svg`—These files display the contents of `celltype_N_anchorfeature_neighborhood.txt` in .png and .svg image formats, respectively.

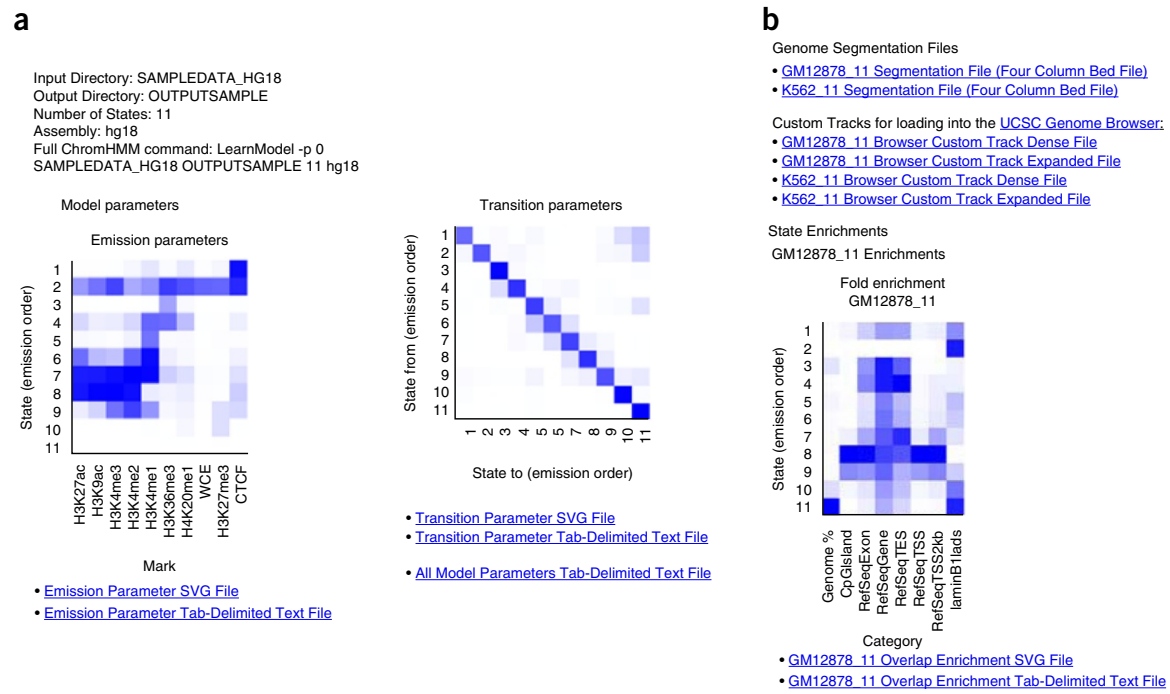

Figure 3 | Example webpage screenshots. The figure displays screenshots of a portion of the webpage automatically generated by the ChromHMM 'LearnModel' command on the sample data. The webpage contains images and links to the files generated by ChromHMM. (**a**) Model parameters. (**b**) Gene segmentation files and state enrichments.

In this command, `inputsegment` specifies the segmentation file for which neighborhood enrichments should be computed. `anchorpositions` specifies the directory containing the files for neighborhood enrichment analysis. These files should be prepared according to Step 9. `outfileprefix` contains the prefix, including possibly the directory, of the output files that will have the extensions `'.txt'`, `'.png'`, and `'.svg'` added to them (**Box 1**).

When executing the commands, also add any appropriate optional parameters. Optional parameters are specified immediately after `NeighborhoodEnrichment`. The full set of options is discussed in the manual. Some options to note are described in **Box 5**.

15| If you desire the states of the model to be in a different order, reorder them with the `Reorder` command by entering

```
java -mx4000M -jar ChromHMM.jar Reorder -o stateorderingfile inputmodel outputdir
```

`inputmodel` is the model file produced by `LearnModel` that should be reordered. `outputdir` is the directory into which the reordered model, and emission and transition parameter files should be written. The option '`-o stateorderingfile`' specifies a file that contains the state reordering. The file is in a two-column, tab-delimited format with the first column containing the old state number and the second column containing the new state number without the letter prefixes. Additional options can be found in the user manual.

16| If the states of the model were reordered in Step 15, then to generate a segmentation corresponding to the reordered states or to generate a segmentation for data not used in learning an existing model, but in the same format, apply the `MakeSegmentation` command by entering

```
java -mx4000M -jar ChromHMM.jar MakeSegmentation modelfile inputdir outputdir
```

In the above `modelfile` is the file containing the model for which segmentation files should be produced. `inputdir` specifies the directory containing the binarized input data. `outputdir` specifies the directory into which the segmentation files should go.

17| If you desire a change of state colors in the browser files, use the `MakeBrowserFiles` command by entering

```
java -mx4000M -jar ChromHMM.jar MakeBrowserFiles -c colormappingfile segmentfile
segmentationname outputfileprefix
```

In the above, `segmentfile` is the segmentation file in a four-column BED format. `segmentationname` is a name for the segmentation. `outputfileprefix` is the prefix including a directory for the regenerated browser files. The extensions `'_browserexpanded.bed'` and `'_browserdense.bed'` are appended to `outputfileprefix`.

When executing the commands, also add any appropriate optional parameters. Optional parameters are specified immediately after the `MakeBrowserFiles` command. The full set of options is discussed in the manual. The option noted above: '`-c colormappingfile`', specifies a file containing the state color mapping. The file is in a two-column, tab-delimited format. The first column gives the state number without a prefix, and the second column gives the color in RGB format, with comma-delimited values between 0 and 255. Choosing chromatin-state colors that are consistent with widely used chromatin-state models can facilitate the interpretation of visualizations of state annotations (e.g., refs. 1,3,12 and **Fig. 1b**).

? TROUBLESHOOTING

Troubleshooting advice can be found in **Table 2**.

TABLE 2 | Troubleshooting table.

Step	Problem	Possible reason	Solution
7A(i), 7B(i)	There are no present calls (1 value) in the binarized data	Peak calls were provided, as opposed to aligned reads, and the '-peaks' flag was not specified	Use aligned reads or add the '-peaks' option
10	Error message that the information initialization strategy can support only a certain number of states	The binarization might not have been done correctly, and there are no present calls. Alternatively, more states are being asked for than there are observed combinations of marks	If there are no present calls, see troubleshooting comments for Step 7. If there are more states than combinations of marks, either reduce the number of states or instead use the random initialization option (**Box 3**)
	Error message indicating insufficient memory	Java does not have access to enough memory or there is insufficient memory available on the system	Increase the memory Java has access to by increasing the value associated with the '-mx' flag. Alternatively, if you are using the '-p' flag, set it to a small nonzero integer value to reduce the number of processors, and thus memory used simultaneously, at a cost of additional run time

● TIMING

Steps 1–3, installation: 5 min
Steps 4–7, binarization: 1 h
Steps 8 and 9, file preparation for enrichment analysis: 30 min
Steps 10 and 11, model learning: typically ~4 h, but can be highly variable
Steps 12–17, post-model-learning analysis: 2 h

ANTICIPATED RESULTS

The expected output of ChromHMM through the main '`LearnModel`' command consists of a number of different files summarized in **Box 6**. These files are all linked to an automatically generated webpage report file (**Fig. 3**). The outputs of other specific commands of ChromHMM are described in **Box 1**.

ACKNOWLEDGMENTS We acknowledge the ENCODE and Roadmap Epigenomics consortia for generation and processing of data to which we have previously applied ChromHMM. We acknowledge the users of ChromHMM who have provided useful feedback on the software. We acknowledge funding from U.S. National Institutes of Health grants U54HG004570, RC1HG005334 (M.K.), R01ES024995, U01HG007912 and U01MH105578 (J.E.); a U.S. National Science Foundation Postdoctoral Fellowship (0905968) and CAREER Award 1254200 (J.E.); and an Alfred P. Sloan Fellowship (J.E.).

AUTHOR CONTRIBUTIONS J.E. and M.K. wrote this protocol and previously developed ChromHMM.

COMPETING FINANCIAL INTERESTS The authors declare no competing financial interests.

1. Ernst, J. *et al.* Mapping and analysis of chromatin state dynamics in nine human cell types. *Nature* **473**, 43–49 (2011).
2. Maurano, M.T. *et al.* Systematic localization of common disease-associated variation in regulatory DNA. *Science* **337**, 1190–1195 (2012).
3. Roadmap Epigenomics Consortium. Integrative analysis of 111 reference human epigenomes. *Nature* **518**, 317–330 (2015).
4. Claussnitzer, M. *et al.* FTO obesity variant circuitry and adipocyte browning in humans. *N. Engl. J. Med.* **373**, 895–907 (2015).
5. Barski, A. *et al.* High-resolution profiling of histone methylations in the human genome. *Cell* **129**, 823–837 (2007).
6. Zhang, Y. *et al.* Model-based analysis of ChIP-Seq (MACS). *Genome Biol.* **9**, R137 (2008).
7. Ernst, J. & Kellis, M. Discovery and characterization of chromatin states for systematic annotation of the human genome. *Nat. Biotechnol.* **28**, 817–825 (2010).
8. Ernst, J. & Kellis, M. ChromHMM: automating chromatin-state discovery and characterization. *Nat. Methods* **9**, 215–216 (2012).
9. Wang, Z. *et al.* Combinatorial patterns of histone acetylations and methylations in the human genome. *Nat. Genet.* **40**, 897–903 (2008).
10. ENCODE Project Consortium. An integrated encyclopedia of DNA elements in the human genome. *Nature* **489**, 57–74 (2012).
11. Hoffman, M. *et al.* Integrative annotation of chromatin elements from ENCODE data. *Nucleic Acids Res.* **41**, 827–841 (2013).
12. Ernst, J. & Kellis, M. Large-scale imputation of epigenomic datasets for systematic annotation of diverse human tissues. *Nat. Biotechnol.* **33**, 364–376 (2015).
13. Mortazavi, A. *et al.* Integrating and mining the chromatin landscape of cell-type specificity using self-organizing maps. *Genome Res.* **23**, 2136–2148 (2013).
14. Chronis, C. *et al.* Cooperative binding of transcription factors orchestrates reprogramming. *Cell* **168**, 442–459 e20 (2017).
15. Javierre, B.M. *et al.* Lineage-specific genome architecture links enhancers and non-coding disease variants to target gene promoters. *Cell* **167**, 1369–1384 e19 (2016).
16. Lorzadeh, A. *et al.* Nucleosome density ChIP-Seq identifies distinct chromatin modification signatures associated with MNase accessibility. *Cell Rep.* **17**, 2112–2124 (2016).
17. Yue, F. *et al.* A comparative encyclopedia of DNA elements in the mouse genome. *Nature* **515**, 355–364 (2014).
18. Roy, S. *et al.* Identification of functional elements and regulatory circuits by *Drosophila* modENCODE. *Science* **330**, 1787–1797 (2010).
19. Rosenbloom, K.R. *et al.* ENCODE data in the UCSC Genome Browser: year 5 update. *Nucleic Acids Res.* **41**, D56–D63 (2013).
20. Cunningham, F. *et al.* Ensembl 2015. *Nucleic Acids Res.* **43**, D662–D669 (2015).
21. Denholtz, M. *et al.* Long-range chromatin contacts in embryonic stem cells reveal a role for pluripotency factors and polycomb proteins in genome organization. *Cell Stem Cell* **13**, 602–616 (2013).
22. Core, L.J. *et al.* Analysis of nascent RNA identifies a unified architecture of initiation regions at mammalian promoters and enhancers. *Nat. Genet.* **46**, 1311–1320 (2014).
23. Wapinski, O.L. *et al.* Hierarchical mechanisms for direct reprogramming of fibroblasts to neurons. *Cell* **155**, 621–635 (2013).
24. Pope, B.D. *et al.* Topologically associating domains are stable units of replication-timing regulation. *Nature* **515**, 402–405 (2014).
25. Ernst, J. & Kellis, M. Interplay between chromatin state, regulator binding, and regulatory motifs in six human cell types. *Genome Res.* **23**, 1142–1154 (2013).
26. Kheradpour, P. *et al.* Systematic dissection of regulatory motifs in 2000 predicted human enhancers using a massively parallel reporter assay. *Genome Res.* **23**, 800–811 (2013).
27. Hibar, D.P. *et al.* Common genetic variants influence human subcortical brain structures. *Nature* **520**, 224–229 (2015).
28. Gjoneska, E. *et al.* Conserved epigenomic signals in mice and humans reveal immune basis of Alzheimer's disease. *Nature* **518**, 365–369 (2015).
29. De Jager, P.L. *et al.* Alzheimer's disease: early alterations in brain DNA methylation at ANK1, BIN1, RHBDF2 and other loci. *Nat. Neurosci.* **17**, 1156–1163 (2014).
30. Frost, B., Hemberg, M., Lewis, J. & Feany, M.B. Tau promotes neurodegeneration through global chromatin relaxation. *Nat. Neurosci.* **17**, 357–366 (2014).
31. Parker, S.C.J. *et al.* Chromatin stretch enhancer states drive cell-specific gene regulation and harbor human disease risk variants. *Proc. Natl. Acad. Sci. USA* **110**, 17921–17926 (2013).
32. Taberlay, P.C., Statham, A.L., Kelly, T.K., Clark, S.J. & Jones, P.A. Reconfiguration of nucleosome-depleted regions at distal regulatory elements accompanies DNA methylation of enhancers and insulators in cancer. *Genome Res.* **24**, 1421–1432 (2014).
33. Al-Tassan, N.A. *et al.* A new GWAS and meta-analysis with 1000Genomes imputation identifies novel risk variants for colorectal cancer. *Sci. Rep.* **5**, 10442 (2015).
34. Lay, F.D. *et al.* Reprogramming of the human intestinal epigenome by surgical tissue transposition. *Genome Res.* **24**, 545–553 (2014).
35. Fiziev, P. *et al.* Systematic epigenomic analysis reveals chromatin states associated with melanoma progression. *Cell Rep.* **19**, 875–889 (2017).
36. Kasowski, M. *et al.* Extensive variation in chromatin states across humans. *Science* **342**, 750–752 (2013).
37. Brown, E.J. & Bachtrog, D. The chromatin landscape of *Drosophila*: comparisons between species, sexes, and chromosomes. *Genome Res.* **24**, 1125–1137 (2014).
38. Day, K. *et al.* Differential DNA methylation with age displays both common and dynamic features across human tissues that are influenced by CpG landscape. *Genome Biol.* **14**, R102 (2013).
39. Horvath, S. DNA methylation age of human tissues and cell types. *Genome Biol.* **14**, 3156 (2013).
40. Ward, L.D. & Kellis, M. HaploReg: a resource for exploring chromatin states, conservation, and regulatory motif alterations within sets of genetically linked variants. *Nucleic Acids Res.* **40**, D930–D934 (2012).
41. Boyle, A.P. *et al.* Annotation of functional variation in personal genomes using RegulomeDB. *Genome Res.* **22**, 1790–1797 (2012).
42. Day, N., Hemmaplardh, A., Thurman, R.E., Stamatoyannopoulos, J.A. & Noble, W.S. Unsupervised segmentation of continuous genomic data. *Bioinformatics* **23**, 1424–1426 (2007).
43. Thurman, R.E., Day, N., Noble, W.S. & Stamatoyannopoulos, J.A. Identification of higher-order functional domains in the human ENCODE regions. *Genome Res.* **17**, 917–927 (2007).
44. Hoffman, M.M. *et al.* Unsupervised pattern discovery in human chromatin structure through genomic segmentation. *Nat. Methods* **9**, 473–476 (2012).
45. Biesinger, J., Wang, Y. & Xie, X. Discovering and mapping chromatin states using a tree hidden Markov model. *BMC Bioinformatics* **14**, S4 (2013).
46. Yu, P. *et al.* Spatiotemporal clustering of the epigenome reveals rules of dynamic gene regulation. *Genome Res.* **23**, 352–364 (2013).
47. Marco, E. *et al.* Multi-scale chromatin state annotation using a hierarchical hidden Markov model. *Nat. Commun.* **8**, 15011 (2017).
48. Roy, S. & Sridharan, R. Chromatin module inference on cellular trajectories identifies key transition points and poised epigenetic states in diverse developmental processes. *Genome Res.* **27**, 1250–1262 (2017).
49. Sohn, K.-A. *et al.* hiHMM: Bayesian non-parametric joint inference of chromatin state maps. *Bioinformatics* **31**, 2066–2074 (2015).
50. Zhang, Y., An, L., Yue, F. & Hardison, R.C. Jointly characterizing epigenetic dynamics across multiple human cell types. *Nucleic Acids Res.* **44**, 6721–6731 (2016).
51. Libbrecht, M.W. *et al.* Joint annotation of chromatin state and chromatin conformation reveals relationships among domain types and identifies domains of cell-type-specific expression. *Genome Res.* **25**, 544–557 (2015).
52. Zacher, B., Lidschreiber, M., Cramer, P., Gagneur, J. & Tresch, A. Annotation of genomics data using bidirectional hidden Markov models unveils variations in Pol II transcription cycle. *Mol. Syst. Biol.* **10**, 768 (2014).
53. Zacher, B. *et al.* Accurate promoter and enhancer identification in 127 ENCODE and roadmap epigenomics cell types and tissues by GenoSTAN. *PLoS ONE* **12**, e0169249 (2017).
54. Mammana, A. & Chung, H.-R. Chromatin segmentation based on a probabilistic model for read counts explains a large portion of the epigenome. *Genome Biol.* **16**, 151 (2015).
55. Song, J. & Chen, K.C. Spectacle: fast chromatin state annotation using spectral learning. *Genome Biol.* **16**, 33 (2015).
56. Duttke, S.H.C. *et al.* Human promoters are intrinsically directional. *Mol. Cell* **57**, 674–684 (2015).
57. Filion, G.J. *et al.* Systematic protein location mapping reveals five principal chromatin types in *Drosophila* cells. *Cell* **143**, 212–224 (2010).
58. Hamada, M., Ono, Y., Fujimaki, R. & Asai, K. Learning chromatin states with factorized information criteria. *Bioinformatics* **31**, 2426–2433 (2015).
59. Jaschek, R. & Tanay, A. Spatial clustering of multivariate genomic and epigenomic information in *Proceedings of the 13th Annual International Conference on Research in Computational Molecular Biology* 170–183 (Springer, 2009).
60. Kharchenko, P.V. *et al.* Comprehensive analysis of the chromatin landscape in *Drosophila melanogaster*. *Nature* **471**, 480–485 (2011).

61. Larson, J.L., Huttenhower, C., Quackenbush, J. & Yuan, G.-C. A tiered hidden Markov model characterizes multi-scale chromatin states. *Genomics* **102**, 1–7 (2013).
62. Roudier, F. *et al.* Integrative epigenomic mapping defines four main chromatin states in *Arabidopsis*: organization of the *Arabidopsis* epigenome. *EMBO J.* **30**, 1928–1938 (2011).
63. Won, K.-J. *et al.* Comparative annotation of functional regions in the human genome using epigenomic data. *Nucleic Acids Res.* **41**, 4423–4432 (2013).
64. Zeng, X. *et al.* jMOSAiCS: joint analysis of multiple ChIP-seq datasets. *Genome Biol.* **14**, R38 (2013).
65. Choi, H., Fermin, D., Nesvizhskii, A.I., Ghosh, D. & Qin, Z.S. Sparsely correlated hidden Markov models with application to genome-wide location studies. *Bioinformatics* **29**, 533–541 (2013).
66. Hon, G., Ren, B. & Wang, W. ChromaSig: a probabilistic approach to finding common chromatin signatures in the human genome. *PLoS Comput. Biol.* **4**, e1000201 (2008).
67. Bernstein, B.E. *et al.* A bivalent chromatin structure marks key developmental genes in embryonic stem cells. *Cell* **125**, 315–326 (2006).
68. Thurman, R.E. *et al.* The accessible chromatin landscape of the human genome. *Nature* **489**, 75–82 (2012).
69. Boyle, A.P. *et al.* High-resolution genome-wide *in vivo* footprinting of diverse transcription factors in human cells. *Genome Res.* **21**, 456–464 (2011).
70. Neph, S. *et al.* An expansive human regulatory lexicon encoded in transcription factor footprints. *Nature* **489**, 83–90 (2012).
71. Ernst, J. *et al.* Genome-scale high-resolution mapping of activating and repressive nucleotides in regulatory regions. *Nat. Biotechnol.* **34**, 1180–1190 (2016).
72. Landt, S.G. *et al.* ChIP-seq guidelines and practices of the ENCODE and modENCODE consortia. *Genome Res.* **22**, 1813–1831 (2012).
73. Kent, W.J. *et al.* The human genome browser at UCSC. *Genome Res.* **12**, 996–1006 (2002).