UNIVERSITY OF COPENHAGEN
FACULTY OF SCIENCE

Master thesis (30 ECTS), MSc in Bioinformatics, Computational Biology

Development of a supervised machine learning tool to infer active regulatory elements from transcription initiation events

Stefano Pellegrini

Supervisor: Robin Andersson

July 4, 2021

ABSTRACT

Transcriptional regulatory elements (TREs) are critical components of gene regulation. These elements, whose mechanisms of action are still not completely understood, provide the main contribution to the regulatory machinery that allows the cells of a multicellular organism to orchestrate their fundamental biological processes. It has been shown that, when active, TREs often overlap open chromatin regions (OCRs) and show characteristic patterns of transcriptional initiation that can be profiled by transcription start site (TSS)-focused 5'-end sequencing techniques.

In this study, we propose a supervised machine learning method that uses TSS profiling data to accurately predict active TREs genome-wide. We trained two predictive models, a gradient boosting based on decision tree and a convolutional neural network, on more than four million profiles extracted by the intersection of data obtained by cap analysis of gene expression (CAGE) and transposase-accessible chromatin using sequencing (ATAC-seq) techniques.

We evaluated the reliability of the method by assessing the model performance on unseen data, obtaining promising results. Ultimately, the final approach was implemented as a user-friendly tool for the genome-wide exploration of active transcriptional regulatory elements in different cell types and tissues.

TABLE OF CONTENTS

INTRODUCTION

Gene regulation is a fundamental process in all life forms that quantitatively and qualitatively control the spatiotemporal expression of genes in the DNA of cells [44] [57]. In multicellular organisms, it is a complex mechanism that allows the execution of essential biological processes, such as differentiation, proliferation, apoptosis, response to environmental stimuli, development and adaptation [44] [55]. Although the synthesis of gene products in eukaryotes can be regulated at each and multiple steps of gene expression, it is more often controlled at the level of transcription [57] [55]. Critical components of transcriptional regulation are transcriptional regulatory elements (TREs), which are genomic elements that can control the transcription of one or more target genes once activated by the binding of transcriptional factors (TF) [87] [55]. Even if significant progress has been made in studying such elements, the understanding of their mechanisms of action is still incomplete [64]. Furthermore, it has been shown that mutation arising in these genomic regions is a major driver of diseases, including cancer [26] [35] [17]. Therefore, the study of TREs is a topic of critical interest in biomedical research.

Historically, these elements have been distinguished in different classes, including promoters, enhancers, insulators, and silencers [26] [41]. Promoters were defined as genomic regions where RNA polymerase II (Pol II) is recruited to locally initiate transcription, while enhancers as elements that can increase the rate of such transcription initiation [17]. Insulators were described as regulatory elements that can act as a barrier, inhibiting enhancer-promoter interactions or the spreading of heterochromatin [34], and silencers as TREs that can repress the transcription of target genes [26]. These classical definitions were based on the questionable assumption that these regulatory elements are distinct classes with different characteristics and molecular functions [17]. This distinction is especially doubtful when focusing on the most characterized TREs having activating functions, i.e., promoters and enhancers [17]. In fact, their classical definition assumed a dichotomous interpretation that recent studies showed to be unsatisfactory [17].

The development of different high-throughput genomic techniques for assessing regulatory activities allowed identifying common and distinct promoters and enhancers' characteristics, attempting their genome-wide identification, and improving the understanding of their functional roles [17] [26]. Transcription start site (TSS)-focused sequencing techniques allowed the profiling of transcription initiation activity of regulatory elements genome-wide [17]. These assays include methods able to capture the 5'-end of steady-state RNA, such as cap analysis gene expression (CAGE), and nascent RNA, such as small 5'-capped RNA sequencing (Start-seq), GRO-cap, and PRO-cap, which are variants of global run-on sequencing (GRO-seq) and precision nuclear run-on sequencing (PRO-seq) [17]. The analysis of data produced by these protocols enabled researchers to show that Pol II initiates transcription at different proximally located loci confined in a nucleosome-depleted region (NDR) [17]. This dispersed transcriptional initiation results in an NDR-confined distribution of core promoters, defined as the TSS proximal regions (±50 base pair (bp)) that include the binding sites for the general transcription factors (GTFs), and therefore determine the selection of the TSSs [17]. Moreover, it has been shown that active promoters and enhancers share a similar promoter architecture, both bidirectionally initiating transcription at the edges of their associated NDR [17]. For example, in promoters, a short and unspliced RNA, namely promoter-upstream transcripts (PROMPTs) or upstream antisense RNAs (uaRNAs), is produced upstream and on the opposite strand of the mRNA transcript, but it is quickly degraded by the nuclear exosome [17]. Likewise, enhancers bidirectionally initiate the transcription of enhancer RNAs (eRNAs), which, similarly to PROMPTs, are short and typically unspliced transcripts and are both quickly degraded by the exosome [17]. Although, it has been shown that, while they can both initiate transcription, transcripts initiated at promoter regions produce more RNA than those initiated at candidate enhancers [17]. Anyhow, the genome-wide detection of divergent and bidirectional TSS became a well-established method for the identification of active regulatory elements [17] [30]. Besides, other established genomic approaches for identifying TREs include assays for genome-wide detection of DNA binding sites for TFs, such as chromatin immunoprecipitation coupled to high-throughput sequencing (ChIP–seq), and assays for genome-wide identification of chromatin accessibility, such DNase I hypersensitive sites sequencing (DNase-seq) and assay for transposase-accessible chromatin using sequencing (ATAC-seq) [26] [31]. ChIP–seq allowed the identification of combinations of histone modifications characterizing the TREs, which were also used to discriminate gene promoters from enhancers, while DNase-seq and ATAC-seq were used to detect candidate TREs by the identification of nucleosome-depleted regions [17] [26]. Unfortunately, the limitation of these genomic techniques is that they aim to detect regulatory elements based on features that do not directly prove their regulatory function [31]. Consequently, TREs identified by these approaches are often inactive [31]. However, two recently developed methods, CRISPR-based in vivo genome editing and massively parallel reporter assays (MPRAs), hold great promise for studying transcriptional regulation [17]. These methods enable the quantification of the promoter and enhancer potential of regulatory elements, defined as the ability to locally initiate transcription and amplify transcription initiation, respectively [17]. This allowed us to improve our understanding of the TREs and their functional roles and made it possible to overcome the dichotomous interpretation of promoters and enhancers, proposing a new model that considers activating TREs as open chromatin regions (OCRs) having different degrees of regulatory activities [17].

In recent years, the increased availability of big data and affordable, powerful computational resources allowed the field

of artificial intelligence, especially machine learning, to become extremely popular [50] [70]. Moreover, the massive amount of data produced by high-throughput techniques led to an exponential growth of the application of machine learning to biological problems [50] [83]. Remarkably, its application to large genomic datasets holds promising results for the annotation and interpretation of functional regulatory elements [54]. For example, recent studies showed that the profiles of transcriptional activities at open chromatin sites can be used for the identification of active TREs through machine learning approaches [31] [86]. However, the potential application of machine learning to TSS-focused sequencing data remains unexplored.

In this study, we propose a supervised machine learning method that uses the profiled transcriptional initiation activity captured by 5'-end sequencing techniques to screen the genome for the detection of active TREs, allowing to investigate different aspects of gene regulation with a single experiment. We trained two predictive models, gradient boosting based on decision tree and convolutional neural network, on a feature vector generated by the intersection of CAGE and ATAC-seq data. We assessed the models' generalization ability by evaluating their performance on unseen data, including data generated from a new cell type and under new conditions, obtaining promising results. Finally, the ultimate approach was used for the development of a user-friendly tool enabling to explore active TREs genome-wide in different cell types and tissues.

Materials and methods

Data

In this study, we integrated data obtained by different high-throughput sequencing assays such as cap analysis of gene expression (CAGE), assay for transposase-accessible chromatin using sequencing (ATAC-seq), and DNase I hypersensitive sites sequencing (DNase-seq). The data integration was finalized to extract labeled feature vectors used to train and evaluate predictive models finalized at the prediction of active transcriptional regulatory elements (TREs).

CAGE: CAGE is a highly precise sequencing-based assay capturing the 5'-end of capped RNA transcripts. It simultaneously detects transcription start sites (TSS) and measures RNA expression levels, making it a suitable method for identifying and studying transcriptional regulatory elements [81]. This technique involves synthesizing cDNA from total RNA, followed by selecting the first 27 nucleotides of the 5'-end of cDNA by the cap-trapping method [81]. Therefore, it allows precise mapping and quantification of DNA tags obtained from the initial nucleotides of the 5'-end RNA transcripts with single-nucleotide resolution [74]. Tags having a common 5' start site are pooled together and are referred to as CAGE-inferred transcription start sites or CAGE-tag starting sites (CTSSs) [25]. Furthermore, proximal CTSSs can be grouped into unidirectional clusters or tag clusters (TCs), if they are located on the same strand, or into bidirectional clusters (BCs), if they include proximal CTSS showing balanced divergent transcription initiation [82]. TCs can be mapped to annotated genes and can be used to estimate gene expression, while BCs can be used to identify enhancer candidates [82].

DNase-seq and ATAC-seq: DNase-seq and ATAC-seq are two high-throughput techniques broadly used to assess genome-wide chromatin accessibility, identify gene regulatory elements, and infer nucleosome positions and transcription factor (TF) binding sites [91] [77] [72] [22]. These techniques are based on cleavage enzymes that recognize and digest DNA regions with increased accessibility. Then, the cleaved fragments are purified, PCR-amplified, sequenced by high-throughput next-generation sequencing, and mapped to a reference genome [53] [77] [23]. DNase-seq uses DNase I to selectively digest OCRs, which are more sensitive to cleavage than regions in closed chromatin state and are referred to as DNase I hypersensitive sites (DHS) [77]. In contrast, ATAC-seq is a recently developed assay that utilizes an engineered hyperactive Tn5 transposase to cleave nucleosome-free DNA regions and insert tag sequencing adapters (a process known as tagmentation) [23] [22]. Compared to DNase-seq, ATAC-seq has the experimental advantage of having a faster and simpler protocol requiring a low amount of input material [22]. Both techniques can be used to infer TF binding sites by a method referred to as computational footprinting [53]. This method relies on the computational scanning of open chromatin profiles to find footprints, small uncleaved regions within larger OCRs that arise due to proteins bound to DNA preventing their enzymatic digestion [53]. However, ATAC-seq footprinting is still poorly explored, and a previous study showed that it obtained lower performance compared to DNase-seq footprinting [53]. A possible explanation is that Tn5 works as a dimer, and it has a complex cleavage mechanism that requires larger binding motifs compared to DNase I, which might prevent Tn5 from binding to accessible DNA of small size [53].

Training, validation, and test sets: To train and perform a preliminary validation of the models, we used in-house CAGE and ATAC-seq data from GM12878, which is a human lymphoblastoid cell line obtained from the blood of a female donor [3]. The Andersson Lab [1] produced these data for an in-house experiment where tumor necrosis factor-alpha (TNFα) was stimulated to induce an inflammatory response used for studying transcription regulation. In this experiment, CAGE and ATAC-seq were obtained at six different timepoints: before the stimulation (timepoint 0), after half-hour (timepoint 0.5), after one hour (timepoint 1), after one and half hour (timepoint 1.5), after two hours (timepoint 2), and after six hours (timepoint 6). For each timepoint, four biological replicates were produced, yielding a total of 24 different replicates. The CAGE data consisted of CTSS datasets in browser extensible data (BED) format, which were trimmed by FASTX-Toolkit [39] and mapped to the hg38 human genome assembly [5] by Burrows-Wheeler aligner [52]. The ATAC-Seq data included significant peaks, defining DNA accessible sites, that were

obtained by MACS2 peak-calling tool [89]. We used both CAGE and ATAC-seq from the first five timepoints (timepoints 0, 0.5, 1, 1.5, and 2) to generate a labeled feature vector used to train, tune, and validate the models by cross-validation. From these data, that will be referred to as training and validation, or *train* and *val*, respectively, we removed chromosomes 2, 3, and 4. In contrast, we selected only chromosomes 2, 3, and 4 from the last timepoint (timepoint 6) of CAGE and ATAC-seq data to generate a test set, which will be referred to as *test-1*.

To further test model performance, we integrated the CAGE data from timepoint 6, again selecting chromosomes 2, 3, and 4, with publicly available DNase-seq data from GM12878 produced by the Meuleman Lab [12] [56], and the resulting test set will be referred to as *test-2*. The Meuleman's data includes high-resolution maps of DHS index obtained from the integration of 733 human biosamples and 438 cell lines using state-of-the-art protocols that have been optimized using an impressive reference panel encompassing a variety of datasets [12] [56].

Furthermore, to evaluate model performance on a different cell line, we used in-house CAGE data obtained from HeLa cells, whose cell line was originally isolated from a cancerous cervical tumor of a human female donor [62]. These data was published in 2014 by Andersson *et al*. [16] to propose a method for the functional classification of RNA based on nuclear stability and transcription initiation profiling. The CAGE data included three control replicates and three treated replicates with knocked out exosome complex component Rrp40 (EXOSC3). We integrated the untreated and treated CAGE replicates with HeLa DHS data from the Meuleman Lab [12] [56], generating two additional test sets that will be referred to as *test-3* and *test-4*, respectively. To perform the integration, since the in-house CAGE data were mapped to the hg19 [4] human genome assembly, we used the liftOver tool from University of California Santa Cruz (UCSC) Genome Browser [47] to lift genomic annotations of the Meuleman DHS data from hg38 [5] to hg19.

The information regarding the training, validation, and test sets mentioned above are summarized in Table I.

Input extraction

Extraction of positive and negative sets: As mentioned, the labeled feature vectors, used to feed the machine learning models finalized at the prediction of active TREs, were generated by performing the intersection between CAGE and ATAC-seq or DNA-seq data.

The input extraction pipeline started by generating the positive set, defined as the CAGE profiles of actively transcribed open chromatin regions (OCRs). First, we extended the central bp of the ATAC-seq peaks or DHS summits of 150 bp in both directions, representing nucleosome-free genomic regions of 301 bp. These were successively filtered by removing overlaps with the problematic loci included in the ENCODE blacklist [15]. Then, we performed the extraction of the CAGE profile of each window, defined as two concatenated vectors including the CTSSs score mapped to each position, on forward and reverse strands, of the selected genomic region. Once we obtained the CAGE profiles of the OCRs, we removed intra-overlaps using a hierarchical approach based on transcriptional activity. This process involved the iterative removal of overlapping regions having the lowest CTSS coverage until all overlaps were solved. Lastly, we filtered the profiles by keeping only those showing a minimum level of transcriptional activity, which we defined as having at least two CTSSs in one of their positions.

The negative set was defined as the CAGE profiles of closed chromatin regions showing a minimum transcription requirement. It was generated by randomly sampling windows of 301 bp from the genome using BEDTools [68]. The sampling was performed avoiding intra-overlaps between the sampled regions, overlaps with the ENCODE blacklist [15], and overlaps with the open chromatin cores, defined as the proximal 100 bp of ATAC-seq peaks or DHS summits. Afterwards, we extracted the CAGE profiles of the selected genomic windows, and, as for the positive set, we filtered them by the minimum requirement of transcription.

Data augmentation: We performed data augmentation to increase the quantity and the quality of the data, and therefore the ability of the models to generalize.

The positive set was augmented by adding the previously defined positive regions shifted by a certain number of bp in both directions. Each replicate was augmented with a different amount of shift to increase variability, as indicated in the following equation.

$$S+ = 10 \cdot R \tag{1}$$

where $S+$ is the amount of bp shift in both directions of the added positive samples, and R is the replicate number. Thus, for example, the positive samples of the training set were augmented by affixing positive profiles moved by ± 10, 20, 30, and 40 bp depending on the replicate number. The idea behind this approach was to add flexibility to the definition of the positive samples. In fact, they were previously defined as the CAGE profiles of the genomic regions centered at the ATAC-seq peaks or DHS summits, while after augmentation, they can be defined as the CAGE profiles of the genomic regions having their center overlapping with an open chromatin core.

Augmentation of the negative set was performed by adding genomic windows flanking the central 101 bp of the OCRs filtered by the hierarchical approach. Successively, we moved upstream or downstream the added windows, depending on their position in respect to the adjacent open chromatin core. As for the positive set, we shifted the windows by a different number of bp, according to the replicate number:

$$S- = \begin{bmatrix} 5 + 5 \cdot R \\ 5 + 5 \cdot R \cdot 2 \\ 5 + 5 \cdot R \cdot 3 \end{bmatrix} \tag{2}$$

where $S-$ is a vector whose elements correspond to the amount of bp shift applied to the added negative samples and, R, as in equation 1, refers to the replicate number.

For example, the negative samples of the training set were augmented by adding three sets of shifted flanking windows: replicate 1 included windows shifted by 10, 20, and 30 bp, while replicate 2 had windows moved by 15, 30, and 45 bp, and so forth. Successively, we removed the added windows overlapping any open chromatin cores. This augmentation approach allowed us to drive the models towards learning the negative samples proximal to the positive ones, which were likely the most difficult to predict.

Processing the profiles: To better capture the patterns of transcriptional initiation, we processed the extracted profiles obtaining the normalized difference of the CAGE signal intensity between the two strands. Each profile was processed as follows:

$$P_{processed} = \frac{f - r}{max(max(f),\ max(r))} \tag{3}$$

where $P_{processed}$ is a vector representing the resulting processed profile, f and r are two vectors whose elements are the CTSS scores mapped to the profile's positions on the forward and reverse strands, respectively. The resulting $P_{processed}$ is the format of the final profiles used to feed the machine learning models, while the previously defined profiles, which we will refer to as the concatenated profiles, were used for visualization purpose and to assess the importance of the CAGE signal of the windows' positions on the individual strands.

Obtaining the final feature vectors: As shown in the Input extraction section, we faced the prediction of active TREs as a binary classification problem. The pipeline to generate the feature vector was implemented to simultaneously extract the samples' profiles and store their metadata information. The metadata included the class labels (positive or negative, encoded as 1 and 0, respectively), chromosomes and window start site coordinates, as well as experimental information, such as replicates number and timepoints information when available. The extraction was performed iteratively one replicate at a time, then, the extracted samples were pulled together obtaining the final datasets shown in Table I.

Model development

Predictive algorithms: In a previous study [11], we already established the feasibility of our approach by using different supervised machine learning algorithms such as random forest, support-vector machine, and gradient boosting based on decision tree (GBDT). In this study, we built predictive models finalized at the classification of active TREs by using the gradient boosting method, which obtained the best performance across the methods tested in the pilot project [11], and the convolutional neural network (CNN) algorithm.

Gradient boosting based on decision tree is an algorithm used to build non-parametric models able to perform both classification and regression [59]. The algorithm allows building a "strong" predictive model that works as a collection or ensemble of "weak" decision trees or base learners [92] [46]. A decision tree is a tree-like structure where each node divides the input data in relation to a particular condition or decision test [78]. Sequentially, each tree is iteratively added to the ensemble in such a way that it is trained concerning the residual error made by the committee of previously added weak learners [59]. The training of each newly added base learner is performed by means of gradient descent, which finds the optimal parameters that minimize a given loss function [59]. The final prediction is obtained by computing the weighted average of the decision trees' votes, where the contribution of each "weak" classifier to the ensemble's prediction depends on its individual performance [59]. In a binary classification problem, the final output of the GBDT is the probability that a given input sample belongs to the positive class ($p(y = 1)$). While, since the two events are mutually exclusive, the probability of the negative class can be obtained as $p(y = 0) = 1 - p(y = 1)$. Finally, the final label can be assigned based on the predicted probability and a certain classification threshold.

In this study, the GBDT models were implemented using Python3 [85] and LightGBM (LGBM) [46] gradient boosting framework. For clarification, we will use the acronyms GBDT and LGBM interchangeably to refer to the gradient boosting algorithm used in this research project.

The convolutional neural network is a special class of artificial neural network (ANN or NN) that has become state-of-the-art in different domains, including computer vision, language, sound, and other signal processing applications [88] [49].

ANN is a non-parametric modeling algorithm that can be described as a directed graph composed of connected units, called artificial neurons or nodes, which can perform regression and classification tasks [40] [90]. The first layer, named the *input layer*, takes the input data and propagates it across the network such that each unit takes as input the output of the previous one [43]. The intermediate layers, called the *hidden layers*, are fully connected (dense) layers composed of one or more nodes that perform a weighted sum of the input tensor (the number of nodes is a layer's hyperparameter). Lastly, the final layer, named the *output layer*, is a fully connected layer that produces the final output [43]. Also, a nonlinear activation function can eventually be applied to the output of the artificial neurons, which allows the network to learn more complex patterns in the data [90]. The training is performed through backpropagation, which uses gradient descent to update the weights and biases of the models to minimize a given loss function [43]. A variant of the gradient descent algorithm, named mini-batch gradient descent, is frequently utilized to increase computational efficiency [80]. This algorithm involves the split of the training data into small batches that are iteratively used to compute the gradient and update the model's parameters [80].

CNN is a flexible and powerful ANN able to detect local spatial features from the input data. It is characterized by features extraction ability and spatial or translation invariance

Feature vector	Cell line	Chromatin assay	Chromosomes	Timepoints	Total replicates	Treatment
Train and val (CV)	GM12878	ATAC-seq	All except 2, 3, 4	0, 0.5, 1, 1.5, 2	20	+ TNFα
Test-1	GM12878	ATAC-seq	2, 3, 4	6	4	+ TNFα
Test-2	GM12878	DNase-seq	2, 3, 4	6	4	+ TNFα
Test-3	HeLa	DNase-seq	All	\	3	\
Test-4	HeLa (Rrp40)	DNase-seq	All	\	3	− Rrp40

TABLE I

Feature vectors summary. The table summarizes the metadata information of the feature vectors generated in this study. Each feature vector was obtained by the intersection between CAGE data and the high-throughput technique used to assess chromatin accessibility indicated in the chromatin assay column. CV in the feature vector names refers to cross-validation used to simultaneously train and validate the models.

[88] [49]. Its typical architecture is composed of multiple elementary units including *convolution*, *pooling*, and *fully connected layers* [88]. The *convolution layer* applies the convolution operation to the input tensor received from the previous node, which is a linear transformation involving applying one or more filters (kernels) whose parameters (weights) are learned during the training process by backpropagation [88]. Each filter can capture a particular local pattern in the input, hence extracting a particular feature, and its application to the input produces a feature map, which is usually processed by a nonlinear activation function [88]. The application of a filter involves an element-wise multiplication between its values and the input, followed by the sum of the resulting values [88]. The operation is performed in a sliding fashion, letting the filter to stride over the positions of the input until the feature map is completed [88]. Hyperparameters of the *convolution layer* include the filters stride, referring to the number of positions shifts over the input, the number and size of the filters, and the padding (zero padding), which allows creating a "border effect" enabling the detection of features at the edges of the input [88]. Typically, a *convolution layer* is followed by a *pooling layer*, but many different architectural variations exist [88] [73]. The *pooling layer* includes a filter with no trainable parameters, performing a pooling operation such as max, average, global max, and global average pooling. It performs the downsampling of the feature maps by taking the average or maximum values of their regions covered by the filter (max and average pooling) or the average or maximum values of the whole feature maps (global max and global average pooling) [88]. The pooling operation ensures spatial invariance of the extracted features and reduces the trainable network's parameters [88]. As in the *convolution layer*, hyperparameters of the *pooling layer* include the filter size, filter stride, and padding [88].

As mentioned, activation functions are often used to process the output of the nodes. One of the most popular activation functions used to process the output of intermediate layers is the rectified linear activation (ReLU) [63]. ReLU, as shown in equation 4, simply directly outputs the input if positive and zero if negative.

$$ReLu(X) = \begin{cases} 0 & x < 0 \\ x & x \geq 0 \end{cases} = \max(0, x) \tag{4}$$

ReLU's popularity is due to the fact that it helps the network overcome the vanishing gradient problem, which prevents the weights from being effectively updated [63]. An activation function can also follow the fully connected *output layer*. In classification problems, which require the network to output the target classes' probabilities, typically, a sigmoid or softmax function is used, depending on the task [79]. The sigmoid function (equation 5), used as the nonlinear final activation function in binary classification problems (where the two classes are encoded as 0 and 1), has a typical "S" shape, and it squashes any real number taken as input into a number between 0 and 1 [79]. As shown in equation 6, the function's output corresponds to the probability of the input belonging to the positive class (class 1), and, since the classes are mutually exclusive, the probability of the negative class (class 0) can be obtained as shown in equation 7 [88].

$$\text{sig}(x) = \frac{1}{1 + e^{-x}} \tag{5}$$

$$p(y = 1) = \text{sig}(x) \tag{6}$$

$$p(y = 0) = 1 - p(y = 1) = 1 - \text{sig}(x) \tag{7}$$

where $\text{sig}()$ is the sigmoid function, x is the input, and y is the true class of x. The softmax function is used in multiclass classification problems [79]. It takes as input a vector of K real numbers, where K is equal to the number of classes, and it outputs a vector of length K whose sum equals 1 [79]. Therefore, as shown in the following equation, it maps the output of the final layer to the probability distribution of the target classes [79].

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \tag{8}$$

where $\sigma()$ is the softmax function, z is the feature vector, and K is the number of classes [79]. Independently from the final activation functions, the final predicted labels are assigned based on the predicted probabilities and a selected classification threshold as in most classification algorithms.

The CNN models implemented in this study were built using Python3 (version 3.8.5) [85], TensorFlow (version 2.2.0) [14], and Keras (version 2.4.3) [29].

Loss function: In predictive analytics, a loss function or cost function is a function used to evaluate the performance of a predictive model by computing a measure of the error obtained by its predictions [61]. During the training of a machine learning model, an optimization algorithm (gradient descent for both CNN and GBDT) is used to find the optimal model parameters that minimize a given loss function [61]. We used cross-entropy for both predictive algorithms used in this study, which is a widely used loss function for classification tasks [65]. In a binary classification problem, where the two target classes are labeled as 0 and 1, the binary cross-entropy or log-loss function can be computed as the average cross-entropy across the predicted data points:

$$\mathrm{BCE} = -\frac{1}{N}\sum_{n=i}^{N} y_i \log\left(p\left(y_i\right)\right) + \left(1-y_i\right)\log\left(1-p\left(y_i\right)\right) \tag{9}$$

where N is the number of predicted samples, y_i is the true class or ground truth of sample i, and $p(y_i)$ is the predicted probability of the samples i belonging to the true class y_i [58] [38].

Regularization techniques: Regularization methods are different strategies applied in machine learning that aim to prevent the overfitting of the training data and, therefore, increase the predictive model's generalization ability [71] [24] [59]. Deep learning and gradient boosting models are both prone to overfitting, but we used two simple and effective techniques to overcome this problem: early stopping and dropout.

Early stopping, which we used in both GBDT and CNN, is a regularization method that can use various stopping criteria to avoid overfitting [33]. A widely used stopping criterion, which we applied to the models trained in this study, is to monitor the training progress and arrests it once the generalization performance (model performance on the validation data) does not show any improvement for a certain number of consecutive training iterations [33] [59]. After arresting the training, the model that obtained the best generalization performance (lowest cross-entropy loss on the validation data) was loaded, saved for future application, and used to perform the predictions.

Dropout is a regularization technique used in ANNs [18]. It reduces overfitting by training a neural network using a combination of different architectures obtained by randomly removing specific nodes (by setting their weights to zero) with a given probability (dropout rate) [67].

In GBDT we also used L2 and L1 regularization. In gradient boosting models, these regularization terms add a penalty to the weights such that: L2 regularization penalizes weights that are too large, while L1 regularization penalizes the non-zero parameters [84].

Training and validation: As previously explained in the Data subsection, the models used in this study were trained on the training set generated from the intersection of CAGE and ATAC-seq data obtained from the GM12878 cell line at different timepoints before and after TNFα stimulation. Simultaneously, the same feature vector was also used to validate the performance using cross-validation (CV). The cross-validation was performed in such a way as to divide the data into training and validation according to chromosomes. The splits were performed iteratively and randomly (using a random seed to ensure reproducibility and comparability between results), assigning 21 chromosomes to the training set and the remaining three chromosomes to the validation set, therefore performing 7-folds cross-validation. Since we used imbalanced datasets, after splitting the data into training and validation, each CV iteration started by performing class weighting as indicated in TensorFlow documentation [21]:

$$w_P = \frac{1}{n_P(n_P+n_N)} \tag{10}$$

$$w_N = \frac{1}{n_N(n_P+n_N)} \tag{11}$$

where w_P and w_N are the weights of the positive and the negative classes, respectively, while n_P and n_N are the numbers of positive and negative samples, respectively. Then, a pre-processor (`StandardScaler()`) from the Scikit-learn library [66], which standardizes the input to zero mean and one unit variance, was fitted to the training data, saved, and applied to both training and validation data. Next, depending on the selected algorithm, a boosting (GBDT) or deep learning model (CNN) was trained, saved, and evaluated, showing cross-entropy loss, F1-score, precision, and recall both as static measures and as a progressive change of the metrics during the training process. Importantly, in both CNN and GBDT, we used early stopping to reduce overfitting. Finally, the predictions performed at each CV step were stored and used to evaluate the average model performance across all iterations.

Therefore, the function performing the training and validation process was implemented to use CV to train, save, and evaluate k models (7 by default), whose number depends on the number of chromosomes used as validation, which determines the number of CV k-folds. Furthermore, it also outputs a data frame including the true classes, the predicted labels, and the inferred probabilities of both training and validation samples predicted across all iterations. Lastly, if the GBDT algorithm is selected, the training pipeline also computes and outputs the average feature importance across the CV iterations. The feature importance is computed by the LightGBM package [46] as the average information gain obtained by a certain feature when it is used in the decision trees [8].

GBDT hyperparameter tuning: We used cross-validation and Bayesian optimization to find the near-optimal set of hyperparameters of the gradient boosting model. To clarify, we refer to hyperparameters as the model parameters that are not updated during the training process but must be set in advance.

Hyperparameter	Tuned value
Num iterations	3000
Early stopping rounds	500
Learning rate	0.02615
Num leaves	1615
Max depth	14
Min data in bin	43
Min data in leaf	2
Min gain to split	1.853
Min child weight	3.1
Feature fraction	0.71
Bagging fraction	0.96
Lambda l1	0.0241
Lambda l2	4.907

TABLE II
Gradient boosting hyperparameters. The table shows the hyperparameters of the tuned GBDT model. *Num iterations* and *early stopping rounds* have been manually adjusted by trial-and-error, while the rest of the hyperparameters have been tuned using Bayesian optimization.

Bayesian optimization aims to find the extrema of an unknown objective function by iteratively using a prior probability distribution to generate a posterior probability distribution of a surrogate function, which approximates the objective one [37] [2] [76]. To perform Bayesian optimization, we used the BayesianOptimization Python package [2], which uses the Gaussian Process as surrogate function to perform optimization, and we choose the F1-score as the objective function to maximize during the process. Furthermore, we choose the hyperparameters space defined as the set of hyperparameters and their bounds.

The final hyperparameters of the tuned boosting model are shown in Table II. The first two hyperparameters, *num iterations* and *early stopping rounds*, were manually set by a simple trial-and-error technique. In contrast, the other hyperparameters shown in Table II were tuned using the BayesianOptimization package [2]. According to one of LightGBM's developers [7] [46], the most important hyperparameters to set are: *num iterations*, which determines the number of boosting iterations, *early stopping rounds*, which sets the number of allowed training iterations without improvement, *learning rate*, which determines the step size for updating the weights during gradient descent, *num leaves*, which sets the maximum number of leaves in one decision tree, and *max depth*, which determines the maximum ensemble trees' depth [10] [6] [9]. Furthermore, as explained in the Regularization techniques subsection, we used L1 and L2 regularization methods by tuning *lambda l1* and *lambda l2* hyperparameters.

***CNN architecture and hyperparameter tuning**:* Starting from a simple baseline ANN architecture, we tuned and built several NN architectures by a simple trial-and-error approach, where each new model was evaluated and compared to the previous best-performing one (best F1-score on the validation data). A summary table showing some of the neural network architectures is shown in Table III. The total number of trainable parameters (weights and biases) of the four architectures shown in Table III are 10,209 (*ANN Baseline*), 30,881 (*CNN-1*), 43,297 (*CNN-2*), and 292,065 (*CNN-Final*).

With the same approach, we also manually tuned the *learning rate*, *early stopping rounds*, *number of epochs*, *batch size*, *dropout rate*, convolutional *filter size*, *filter stride* and *padding*, and max and global max *pooling size*, which each was set to a common value among all architectures shown in Table III (e.g. the *pooling size* was set to 5 for all pooling layers of each architecture). These hyperparameters are shown in Table IV.

Also, in all models, we used the ReLU function (equation 4) as activation function on the intermediate layers (convolutional and dense layers), and the sigmoid function (equation 5) as activation function on the output layer. Lastly, all NN models were trained using the Adam optimizer [48] (*learning rate* shown in Table IV).

***Prediction of new data**:* We implemented the function to predict new data such that it uses the preprocessing parameters and the predictive models (GBDT or CNN) saved during the training process. For each CV iteration used to perform the training, the new data is standardized, and the loaded model is used to perform the prediction. The final prediction is obtained by averaging the probabilities predicted by the models obtained at each CV iteration, and assigning the predicted label according to the classification threshold. If the true labels are provided, the function also evaluates the model performance by generating a heatmap showing some accuracy measures (F1-score, Matthews correlation coefficient, true negative rate, true positive rate, and precision) obtained by the individual models and the ensemble (average predictions across the seven models). The final output of the function is a data frame including the true labels, if provided, and the labels and probabilities predicted by the final ensemble.

***Threshold-moving**:* Threshold-moving is a simple technique that helps to deal with imbalanced datasets [20]. Given a set of predictions, the method involves tuning the binary classification threshold to maximize or minimize a selected metric [20]. We obtained the optimal classification threshold based on the receiver operating characteristic (ROC) curve analysis computed on the predictions obtained on the validation data. The ROC curve is a diagnostic plot that shows the predictive performance of a binary classifier at different classification thresholds, revealing the trade-off between sensitivity and specificity [36]. Given the classification thresholds used to build the ROC curves, for each model, we selected the optimal classification threshold that maximized the Youden's J statistic or Youden's index [75], which is defined as follows:

$$J = \text{sensitivity} + \text{specificity} - 1 \tag{12}$$

$$J = \frac{\text{TP}}{\text{TP} + \text{FN}} + \frac{\text{TN}}{\text{TN} + \text{FP}} - 1 \tag{13}$$

ANN Baseline			
Layer type	**Nodes/Filters #**	**Output shape**	**Par #**
Dense	32	(None, 32)	9664
Dense	16	(None, 16)	528
Dropout	0	(None, 16)	0
Dense	1	(None, 1)	17
CNN-1			
Layer type	**Nodes/Filters #**	**Output shape**	**Par #**
Conv1D	32	(None, 301, 32)	128
MaxPooling1D	0	(None, 60, 32)	0
Flatten	0	(None, 1920)	0
Dense	16	(None, 16)	30736
Dropout	0	(None, 16)	0
Dense	1	(None, 1)	17
CNN-2			
Layer type	**Nodes/Filters #**	**Output shape**	**Par #**
Conv1D	32	(None, 301, 32)	128
Conv1D	32	(None, 301, 32)	3104
Conv1D	32	(None, 301, 32)	3104
Conv1D	32	(None, 301, 32)	3104
Conv1D	32	(None, 301, 32)	3104
MaxPooling1D	0	(None, 60, 32)	0
Flatten	0	(None, 1920)	0
Dense	16	(None, 16)	30736
Dropout	0	(None, 16)	0
Dense	1	(None, 1)	17
CNN-Final			
Layer type	**Nodes/Filters #**	**Output shape**	**Par #**
Conv1D	32	(None, 301, 32)	128
Conv1D	32	(None, 301, 32)	3104
Conv1D	32	(None, 301, 32)	3104
Conv1D	32	(None, 301, 32)	3104
Conv1D	32	(None, 301, 32)	3104
MaxPooling1D	0	(None, 60, 32)	0
Conv1D	64	(None, 60, 64)	6208
Conv1D	64	(None, 60, 64)	12352
Conv1D	64	(None, 60, 64)	12352
Conv1D	64	(None, 60, 64)	12352
Conv1D	64	(None, 60, 64)	12352
MaxPooling1D	0	(None, 12, 64)	0
Conv1D	128	(None, 12, 128)	24704
Conv1D	128	(None, 12, 128)	49280
Conv1D	128	(None, 12, 128)	49280
Conv1D	128	(None, 12, 128)	49280
Conv1D	128	(None, 12, 128)	49280
GlobalMaxPooling1D	0	(None, 128)	0
Dense	16	(None, 16)	2064
Dropout	0	(None, 16)	0
Dense	1	(None, 1)	17

TABLE III

Neural network architectures. The table shows three neural network architectures that we evaluated in a trial-and-error approach to obtain the final model, and the architecture of the final model itself (*CNN-Final*). Each row includes summary information of a specific layer, and each layer takes as input the output of the previous one. From left to right, the columns specify the following information: layer type, number of nodes or filters, output shape, and number of trainable parameters.

Hyperparameter	**Tuned value**
Learning rate	0.0001
Early stopping rounds	10
Number of epochs	200
Batch size	2048
Conv1D filter size	3
Conv1D filter stride	1
Conv1D padding	zero padding
Pooling size	5
Dropout rate	0.5

TABLE IV

Neural network hyperparameters. The table shows the hyperparameters of the models built using the architectures shown in Table III, including the final CNN model. These hyperparameters, as well as all the NN hyperparameters, were manually tuned using a trial-and-error approach. Hyperparameters specific to a particular layer (e.g. *Conv1D filter size* or *Pooling size*) were shared by all layers of that type in all models.

, where TP, FN, TN, and FP refer to the number of true positive, false negative, true negative, and false positive predicted samples [19], respectively.

***Performance evaluation**:* We generated a variety of plots showing the performance of the models trained and applied to the different feature vectors generated in this study (Table I). First, as a diagnostic tool, we generated plots showing the progressive change of the cross-entropy loss, F1-score, precision, and recall during the training process. Particular care was taken to create the diagnostic plots for the deep learning models, generated using a custom Keras callback [27] that allowed us to store the real performance obtained on both the training and validation data at the different epochs of the training process. In contrast to our method, the default method used by Keras stores the training history of the model, evaluating the performance on the training data as a running average over the batches of a given epoch [28]. At the same time, the performance obtained on the validation data is evaluated at the end of each epoch [28]. Furthermore, in the built-in method used by Keras [27], dropout is active while evaluating the model on the training data, but it is inactive on validation, making the comparison between training and validation performance even more difficult [28]. Also, we used the ROC curve, precision-recall curve, confusion matrix, and a summary heatmap showing F1-score (macro average), Matthews correlation coefficient (MCC), true negative rate (TNR), true positive rate (TPR or recall), and precision obtained by the gradient boosting and the deep learning models on the different datasets. Furthermore, we analyzed the probability distribution, the average CAGE profiles, and the principal component analysis (PCA) plots of the predicted samples, considering confusion matrix information (TP, TN, FP, and FN predictions).

Tool and genome-wide prediction

The tool we implemented in this study allows the user to perform a genome-wide prediction of active TREs from CAGE data using trained gradient boosting or CNN models. The pipeline takes as input a CTSS file, and it scans the genome extracting 301 bp long CAGE profiles of regions having a minimum level of transcriptional activity (two CTSS in at least one position). To reduce computational complexity, the scan starts and ends at the first and the last transcribed loci of each chromosome and, by default, is performed with steps of 5 bp. Next, it predicts the probabilities of the extracted profiles to be potential active TREs, and it generates different diagnostic plots, including the average shape of the CAGE signal of the predicted samples and their distribution across the chromosomes. The final output includes a BED file that can be directly loaded into the UCSC Genome Browser [47] for interactive visualization.

Input and output: Our tool takes a CTSS file as input and it generates five tabular output files and different plots: two files including the CAGE profiles of the extracted regions (`profiles_<filename>.csv`, which are the concatenated profiles and `profiles_<filename>_subtnorm.csv`, which are the processed ones), one file including metadata information (`metadata_<filename>.csv`) such as the genomic coordinates, one file including the predictions (`tres_prediction_<filename>.csv`), and one BED file including the predicted scores ready to be visualized as UCSC Genome Browser [47] track (`tres_prediction_UCSC_track_<filename>.bed`). Additional information about the usage of the tool can be found in Supplementary materials.

RESULTS

Input exploratory analysis

We used the intersection between CAGE and ATAC-seq or DNase-seq data to generate different feature vectors used to train and evaluate supervised machine learning models for the prediction of active transcriptional regulatory elements (TREs). We defined two classes of genomic regions, a positive class, defined as transcriptional active OCRs, and a negative class, defined as transcriptional active regions in closed chromatin state. The training set was generated from CAGE and ATAC-seq data obtained from the GM12878 cell line, which was collected before and after the stimulation of TNFα at six different timepoints.

Figure 1 shows the average CAGE profiles of positive and negative samples, before and after the augmentation of the training set (timepoint 0). Data augmentation was performed by adding positive samples shifted off a few bp from their original position and negative samples flanking the open chromatin cores, as explained in Materials and methods. This process was performed to increase the quality and quantity of the data, allowing the models to recognize with flexibility the critical patterns of transcriptional initiation observed in OCRs. Also, we wanted the models to focus on learning the profiles of the negative samples close to the positive ones. In fact, these negative samples, compared to the ones sampled randomly from the genome, have a greater transcriptional initiation activity due to the presence of proximal active TREs, and therefore their prediction should be more challenging. It is possible to observe (Figure 1) that the positive samples, both before and after data augmentation, show a characteristic pattern (sigmoid shape) suggesting bidirectional transcription initiation activity centered at nucleosome-depleted regions (NDRs). In contrast, the CAGE signal of the negative samples before data augmentation seems to be almost evenly distributed among their positions, except for the presence of higher signal intensity at the edges of the profiles. Higher signal intensity at the profiles' edges arose because we generated the negative samples allowing partial overlaps with the positive ones. In fact, the central regions of the negative samples, not overlapping with any open chromatin region, display an almost flat CAGE signal. Furthermore, we can see that, as we expected, the signal intensity at the edges of the augmented negative profiles is even more prominent. Also, since the negative samples partially overlap the positive ones, their average profile show a higher intensity of the signal only at the left side, on the forward strand, and only at the right side, on the reverse strand.

Figure 2 shows the average CAGE profiles of four untreated replicates (timepoint 0) of the training set. Recalling that the shift of the added profiles in data augmentation was performed according to the replicates number, we can see that this process added some variability in both positive and negative samples extracted from the different replicates, contributing to increasing the variability of the pulled profiles.

Figure 3 includes the average CAGE profiles of the samples extracted from the individual timepoints of the training set and the concatenated and processed average CAGE profiles of the pulled samples. As expected, we can see that the profiles obtained from the individual timepoints show very similar but not identical patterns. Also, we can see that the profiles of the pulled samples maintained the characteristic patterns observed in the average profiles obtained from individual replicates and timepoints, but the patterns became progressively more pronounced.

Figure 4 shows the differences between the average CAGE profiles of training samples obtained from different chromosomes. It is possible to observe that overall, the signal of the positive and negative samples is homogeneous among chromosomes, but there are some exceptions. Some chromosomes show particularly high spikes in the signal (e.g., chromosomes 7 in both positive and negative samples), while others show a similar pattern with more noise (e.g., chromosomes 6 and 22 in negative samples) or a completely noisy signal as in chromosomes Y in both positive and negative samples. The spikes of transcriptional initiation are likely generated by genes expressed ubiquitously, which might have similar regulatory processes. In addition, having artifacts on chromosome Y was expected because the data was generated from the GM12878 cell line produced from the blood of a female

Feature vector	Cell line	Chromatin assay	Total profiles #	Positive profiles #	Negative profiles #	Positive profiles %
Train and val (CV)	GM12878	ATaC-seq	4,363,934	2,065,000	2,298,934	47.32%
Test-1	GM12878	ATaC-seq	184,182	88,247	95,935	47.91%
Test-2	GM12878	DNase-seq	84,558	39,803	44,755	47.07%
Test-3	HeLa	DNase-seq	186,015	87,332	98,683	46.95%
Test-4	HeLa (Rrp40)	DNase-seq	301,257	141,436	159,821	46.95%

TABLE V

Feature vectors profiles count. The table shows the number of extracted profiles in each feature vector (Table I) generated in this study. It includes the total count, the count of positive and negative, and the percentage of positive profiles.

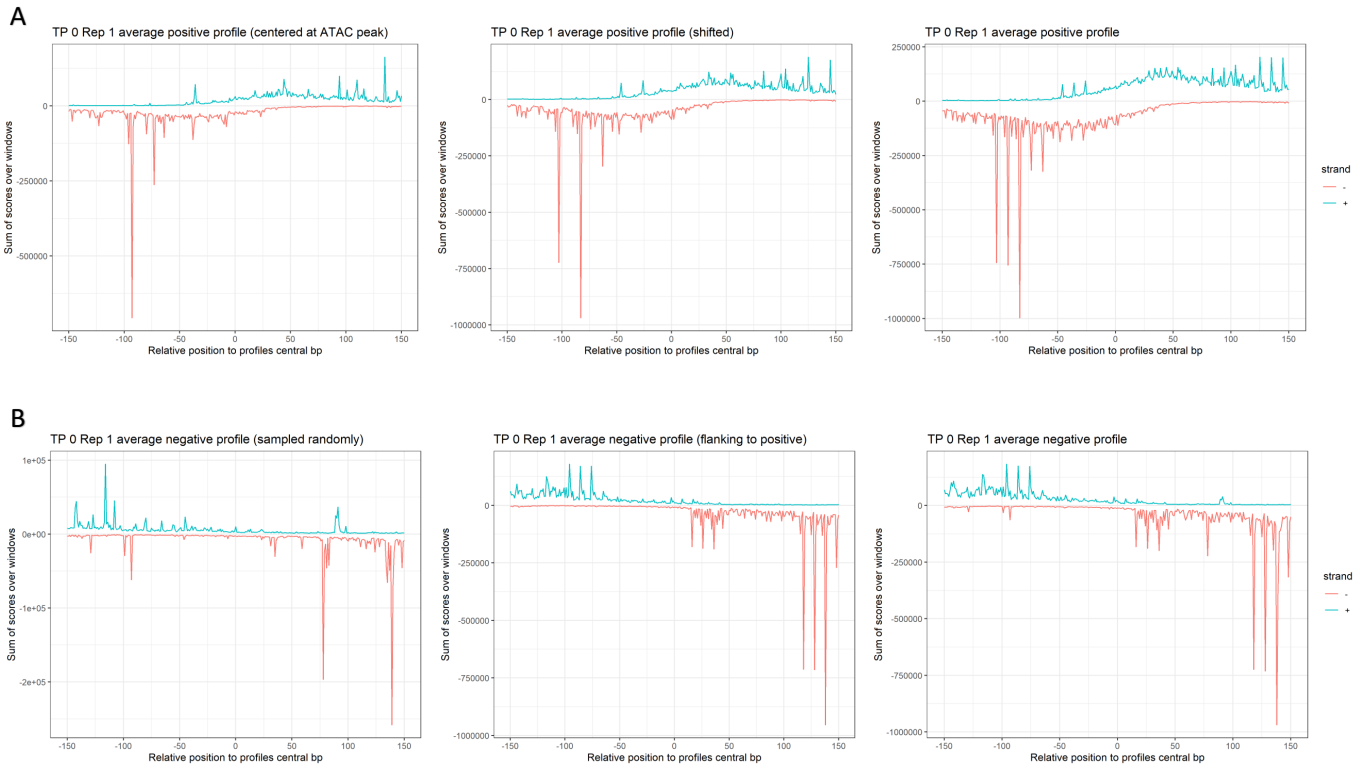

Fig. 1. **Data augmentation, average CAGE profiles of one replicate.** The figure shows the average CAGE profiles of positive (**A**) and negative (**B**) samples of the training set (timepoint 0, replicate 1) before and after data augmentation, performed as described in Materials and methods. Starting from the left, the first column shows the average profiles before data augmentation (obtained from genomic regions centered at the ATAC peaks (**A**) and from negative regions sampled randomly from the genome (**B**)). The second column shows the average profiles of the added samples (positive samples shifted off a few bp in both directions from the central ATAC peaks (**A**), and negative samples flanking the open chromatin cores (**B**)). Finally, the last column shows the profiles of the augmented data. The CAGE average profile was obtained by summing the CTSSs at each position for all positive (**A**) and negative samples (**B**). Therefore, the *y-axis* shows the sum of CTSS scores at a particular window position (across all windows), and the *x-axis* shows the relative window position with respect to its center. For visualization purposes, we assigned a negative value to the CAGE signal of the reverse strand.

donor [3], which therefore lacks the Y chromosome. However, although we could have removed it, the number of profiles extracted from this chromosome is so low (Figure 7) that it should not negatively affect the training and evaluation of the models.

To provide a few examples, Figure 5 shows the concatenated and processed CAGE profiles of four negative and four positive samples of the training set. We can see that all except one positive sample show a bidirectional transcription initiation activity, which is not shown in any of the negative ones. Also, the second positive sample (from left to right), which does not present bidirectionality, and the second negative one, show an example of two different classes having a very similar pattern that might confound the learning process of the models.

To choose the window size for the extraction, we simply visualized the average profiles obtained from genomic regions of different lengths. For example, figure 6 shows the average profiles of non-augmented positive regions (centered at the ATAC peaks) obtained using different lengths: 301, 501, 601, and 1201 bp. We choose the final profile length of 301 bp because, as we can see, at -150 bp and +150 bp, there are two additional shifts in the intensity of the signal between strands, suggesting the presence of flanking nucleosome-depleted regions.

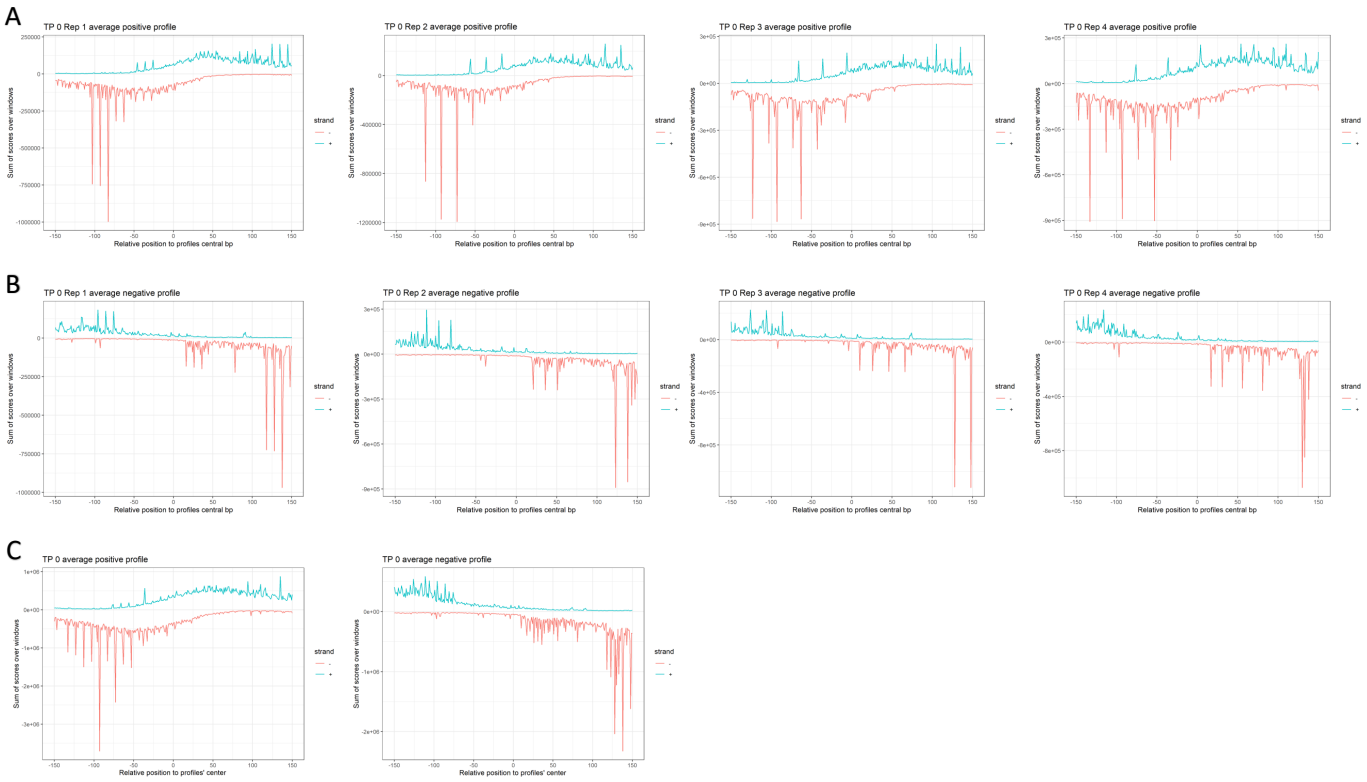

Fig. 2. **Average CAGE profiles of four replicates.** The figure shows the average CAGE profiles (as explained in Figure 1) of positive (**A**) and negative (**B**) samples obtained from the four replicates of the timepoint 0 of the training set. **C** shows the average profiles of the merged positive (left) and negative (right) samples obtained from the four replicates mentioned above.

After performing the profile extraction of the 24 replicates from five different timepoints (Table I), we pulled the resulting profiles obtaining a slightly unbalanced dataset of 4,363,934 training samples: 2,065,000 positives and 2,298,934 negatives (Table V). These profiles were processed to capture the difference in the CAGE signal intensity between strands (Materials and methods), and they were used to train the predictive models. Figure 7 shows the ratio between positive and negative samples of the training set, their distribution among chromosomes, and the difference in their total transcription initiation activity. We can see that the distribution of the two classes is balanced among chromosomes. In addition, as expected, larger chromosomes have a more significant number of actively transcribed samples, and very few samples were extracted from chromosome Y. Looking at the total CTSS scores distribution, it is possible to observe that, compared to the positive samples, the negative ones have more than double the number of profiles with only a basal level of transcriptional initiation (two CTSSs). In contrast, the positive profiles have a greater number of samples with more than 12 total CTSSs. Looking at the boxplot, we can see that a large number of profiles in both classes have a low total transcription initiation activity, with median values close to the minimum requirement. However, the average total CTSS score is much more prominent on the positive samples, having an average of ≈350 CTSSs, than the negative ones, with an average of ≈150.

Table V shows the total number of extracted samples and their proportion between positive and negative classes in each dataset generated in this study. We can see that the training (and validation) set has the largest number of extracted profiles, and all feature vectors have a similar ratio (≈47%) between positive and negative samples.

Model evaluation

After generating the training set, we used it to train and validate different models employing cross-validation (CV). The same dataset was used to perform hyperparameter tuning and model selection. At each CV iteration, we trained one model on the profiles obtained from 21 chromosomes and validated it on the remaining 3 chromosomes. Therefore, the model performance obtained during training and validation is computed as an average across the 7 CV folds.

We trained and evaluated a baseline LGBM model (LGBM-1) with default hyperparameters except for the *learning rate* (0.009), *num iterations* (3000), and *early stopping rounds* (500), and a tuned LGBM model (LGBM-Final), whose hyperparameters are listed in Table II. The tuning of the gradient boosting model was performed using Bayesian optimization. We also trained and tuned different neural networks, whose evaluation allowed us to build the CNN architecture that we used for the final deep learning model (Table III). We used a simple trial-and-error approach to perform the tuning of the NNs and their architecture development. Starting from a

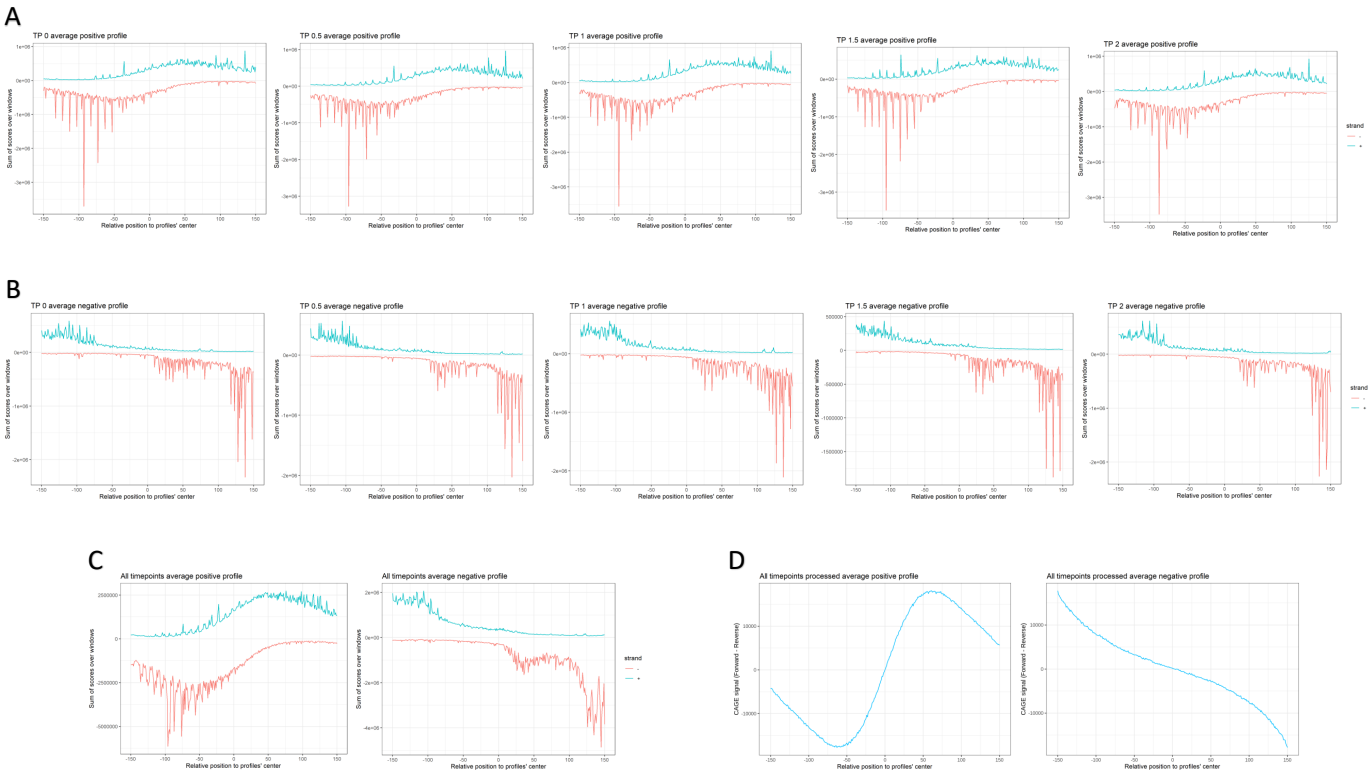

Fig. 3. **Average CAGE profiles of the training set.** The figure shows the positive (**A**) and negative (**B**) average CAGE profiles (as explained in Figure 1) of the individual timepoints of the training set. **C** and **D** show, respectively, the final average concatenated and processed profiles of positive (left) and negative (right) samples of the training feature vector. The concatenated profiles (**C**) were mainly used for visualization purposes, while the processed ones (**D**) were used as input for the machine learning algorithms. The *y-axis* of the processed profiles shows the normalized difference of the CAGE signal intensity between the two strands (Materials and methods).

baseline model (NN-1), we progressively built new neural networks by adding new layers, adjusting their hyperparameters, and evaluating their performance in comparison to the previously trained ones. We iteratively selected the hyperparameters and the architecture of the model that obtained the best generalization performance, defined as the highest F1-score obtained on the validation data (CV). The architectures and hyperparameters of the baseline model (NN-1), the final deep learning model (CNN-Final), and two other neural networks (CNN-1 and CNN-2) that we selected among the ones we implemented to perform model selection are displayed in Tables III and IV. Figure 8 includes the ROC and precision-recall curves showing the performance of the above-mentioned gradient boosting and neural network models used to perform hyperparameter tuning and model selection. Also, a summary including a few accuracy measures (F1-score, MCC, TNR, TPR, and precision) computed on the predictions performed by these models is shown in Table VI. For clarity, we will refer to the performance obtained on the training set as in-sample performance and the performance obtained on the validation set as out-of-sample performance. In both Figure 8 and Table VI, we can observe that, compared to the baseline model, the tuned gradient boosting model significantly improved the in-sample performance, but, while the out-of-sample performance also increased, its change was not so remarkable. The Final-CNN is the best-performing model among the neural networks, obtaining an almost equally good in-sample and out-of-sample performance. In general, we can see that all neural networks showed very low signs of overfitting, and while both in-sample and out-of-sample performance progressively increased from the baseline model to the final CNN, the change was not so conspicuous since the simple baseline NN already obtained satisfactory results. Furthermore, we can see (Table VI) that both algorithms obtained better results on the classification of the negative samples compared to the positive ones. In addition, Table VI also includes the duration of the models' training process, and we can observe that both, the final gradient boosting and deep learning models, increased their training time by almost twofold compared to their baseline counterparts.

Figure 9 shows the training progression of the final models, displaying cross-entropy loss, F1-score, recall, and precision obtained on training and validation data at each training cycle. For visualization purposes, in order to display different metrics, we were forced to disable early stopping on the LGBM model (early stopping on the LGBM algorithm arrests the training progress as soon as one of the evaluation metrics stops improving), but it was still enabled on CNN. As explained in Materials and methods, the deep learning performance was computed using a custom Keras callback

Fig. 4. **Average CAGE profiles of the training set at chromosome resolution.** The figure shows the average CAGE profiles (as explained in Figure 1) of positive (**A**) and negative (**B**) samples of the training feature vector, exhibiting differences between chromosomes.

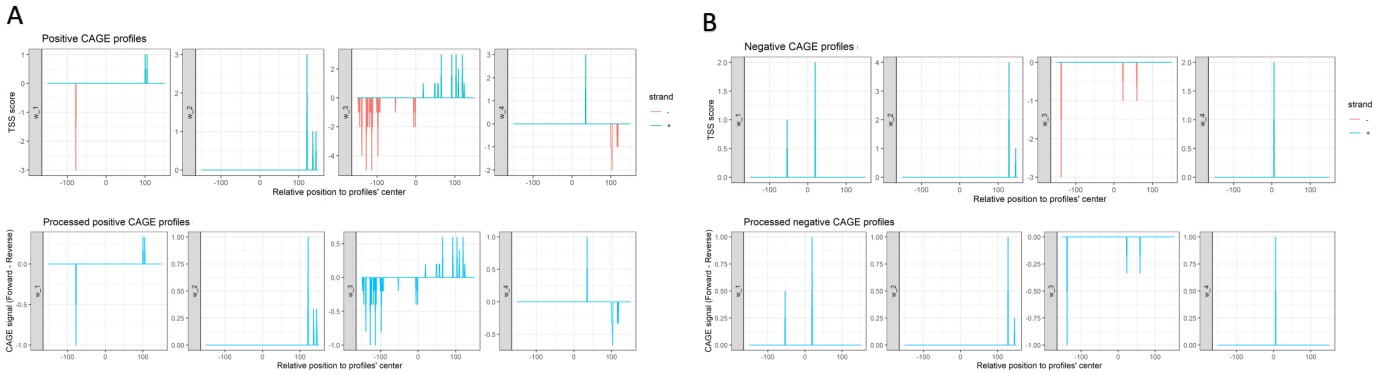

Fig. 5. **Example of four CAGE profiles.** The figure shows the concatenated (top row) and processed (bottom row) CAGE profiles (as explained in Figures 1 and 3) of four positive (**A**) and four negative (**B**) samples of the training set.

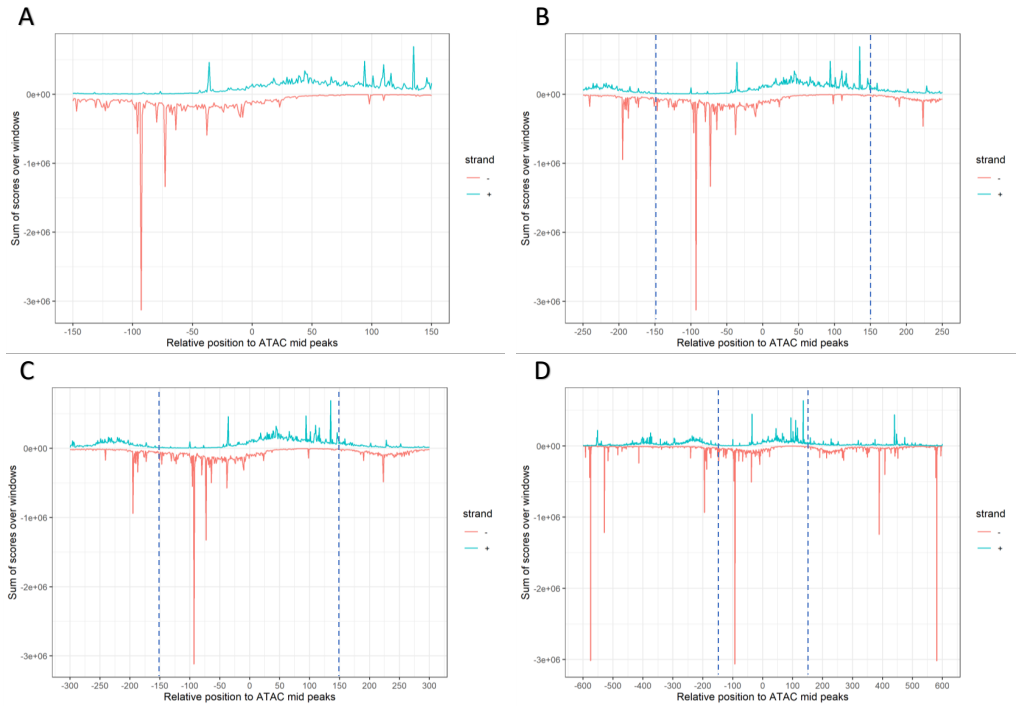

Fig. 6. **Average CAGE profiles length comparison.** The figure shows the average CAGE profiles of the positive samples centered at the ATAC peaks (before data augmentation) obtained using four different lengths: 301 bp (**A**), 501 bp (**B**), 601 bp (**C**), and 1201 bp (**D**). The blue dashed lines mark the length (301 bp as in **A**) that we used to extract the final feature vectors.

[27], which allowed us to compare objectively real in-sample and out-of-sample performance starting from the evaluation of the untrained model (epoch 0). We can observe that the training of the gradient boosting framework was smoother than the deep learning one. However, the optimization process of both algorithms obtained the convergence of the loss function toward global minima values. In addition, we can see that, in all CV iterations, CNN learned the critical patterns in the data very fast, obtaining a notable increase of the performance just in the first epoch.

Figure 10 shows the feature importance computed by the gradient boosting model using as input both the concatenated and processed profiles. For example, we can observe that the model gives more importance to CTSSs approximately located between profiles' positions -70 and -30, on the reverse strand, and between positions +30 and +70, on the forward strand. These roughly match the highest intensity of the signal observed in the average CAGE profiles of the positive samples of the training set (Figure 3 C and D).

To assess the generalization performance, we evaluated the final models on different test sets, whose summary information, as well as information about the training (and validation) set, are included in Tables I and V. *Test-1* feature vector was obtained from the same data used to extract the training set (CAGE and ATAC-Seq data obtained from GM12878 cells [3] at different timepoints before and after the stimulation of

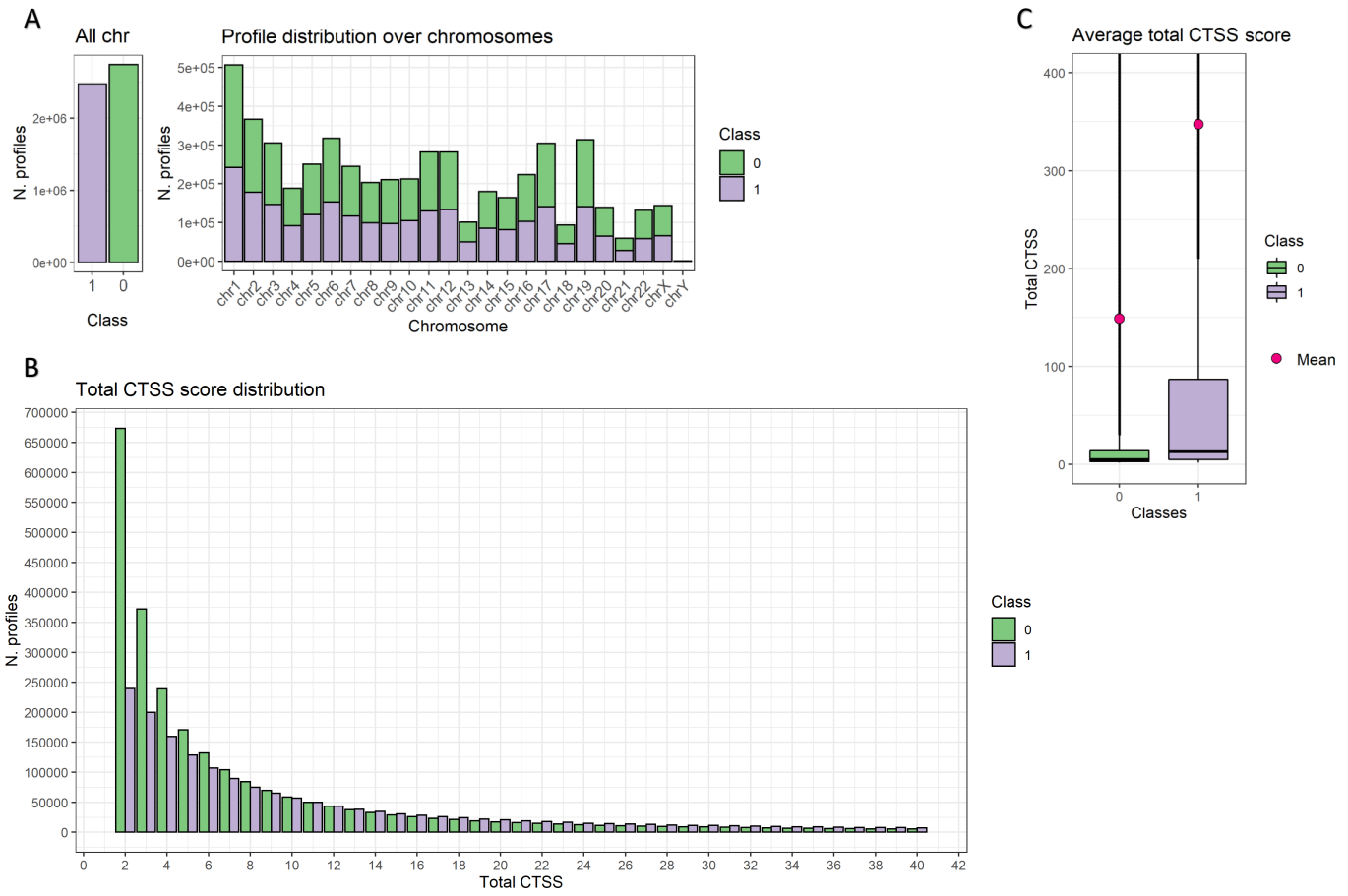

Fig. 7. **Class and total CTSS score distribution of the training set.** The figure shows some exploratory plots regarding the samples of the training set. **A** shows the total number of positive and negative samples (left), and the positive (class 1) and negative (class 0) sample distribution over chromosomes (right). **B** shows the CTSS distribution with class resolution. The *y-axis* shows the count of profiles having a particular total CTSS score, and the *x-axis* shows the total score obtained as a sum of CTSSs over the positions of individual profiles. For visualization purposes, we limited the coordinate of the *x-axis* showing only the number of profiles with a maximum of 40 total CTSSs. **C** shows a boxplot comparing the total transcription initiation activity between positive and negative samples, including the median (black horizontal lines) and mean values (red points).

TNFα), with the major difference that *test-1* was extracted from three chromosomes (chromosomes 2, 3, and 4) and from a different timepoint (six hours after the stimulation) which was not used for the extraction of the training samples. Therefore, this extraction process allowed us to obtain a first test set that was slightly different from the training data, and which offered us the possibility to perform a preliminary assessment of the models' generalization ability. *Test-2* feature vector was generated by the intersection of the CAGE data used in *test-1* with DNase-seq data produced by the Meuleman Lab [12] [56], therefore allowing the evaluation of the model performance on a feature vector generated using an alternative technique for the assessment of chromatin accessibility. To push that even further, we assessed the generalization ability of the models on two additional datasets, *test-3* and *test-4*, that were obtained by the intersection of in-house CTSS data [16] with the Meuleman's DHS data [12] [56] obtained from the HeLa cell line [62]. *Test-4*, differently from *test-3*, was extracted from CAGE data obtained by cells treated by knocking out the exosome complex component Rrp40 (EXOSC3) [16].

Figure 11 shows the confusion matrices, ROC, and precision-recall curves computed on the predictions obtained by the final models on the training, validation, and test sets. In addition, a summary table showing the exact values of the in-sample (performance on the training set) and out-of-sample (performance on the validation and test sets) performance can be visualized in Table VII. In both Figure 11 and Table VII, we can see that, compared to the gradient boosting model, the CNN showed a greater generalization ability, obtaining better out-of-sample performance on each feature vector tested in this study. On the contrary, the LGBM obtained a better result only on the training data, indicating a higher level of overfitting. However, overall, the convolutional neural network and gradient boosting models obtained similar and entirely satisfactory predictions. Also, we can observe (Figure 11 or Figure 17 in Supplementary materials) that, unexpectedly, both algorithms obtained the worst performance on the training and validation set, suggesting that these data likely contained the most challenging profiles to predict. More importantly, it also

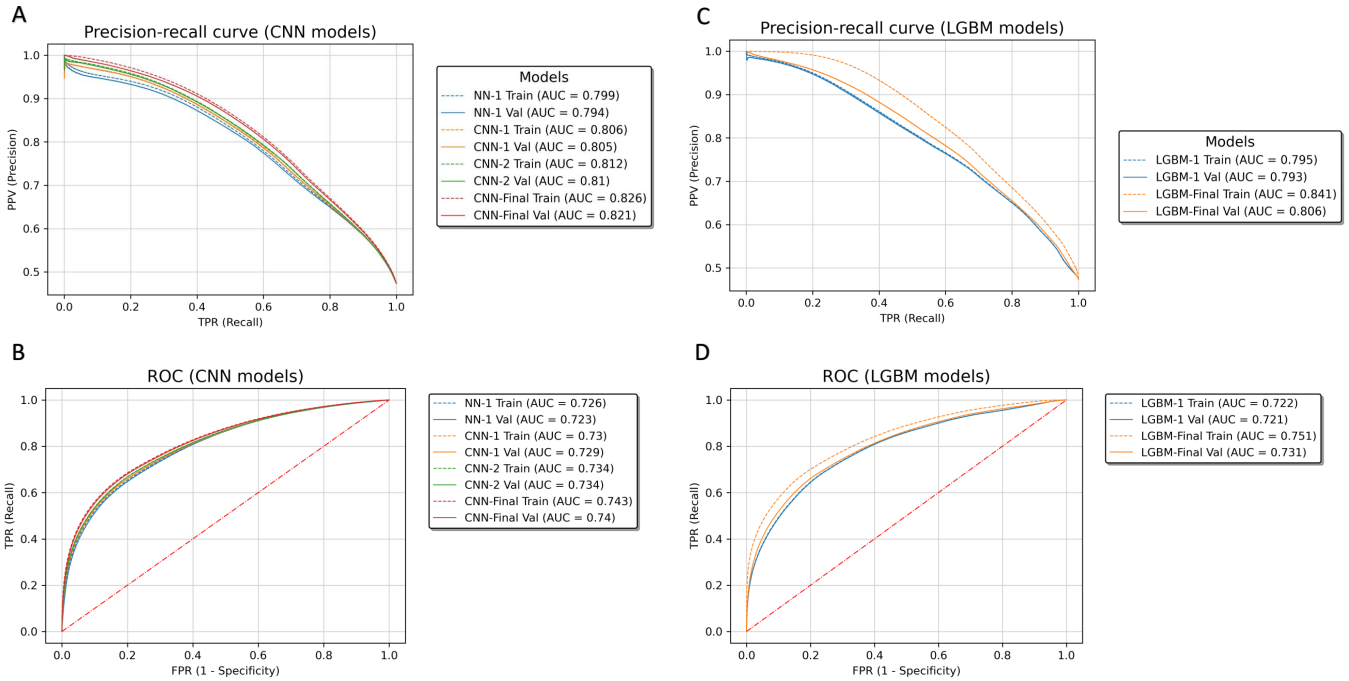

Fig. 8. **Performance comparison for model selection.** The figure shows the training and validation performance of some of the models that we implemented in this study to perform model selection. **A** and **B** show the precision-recall and the ROC curves, respectively, obtained by the neural networks, while **C** and **D** show the precision-recall and the ROC curves obtained by the gradient boosting models. The AUC values in the legends represent the values of the area under the curve they are referring to (ROC and precision-recall). NN-1 and LGBM-1 are the baseline models for the neural network and the LightGBM (gradient boosting) algorithms, while CNN-Final and LGBM-Final are the final models that we used in the tool. The architectures and hyperparameters of the neural networks (NN-1, CNN-1, CNN-2, and CNN-Final) are shown in Tables III and IV, while the hyperparameters of the tuned gradient boosting model (LGBM-Final) are shown in Table II. LGBM-1 used default hyperparameters except for the *learning rate* that was set to 0.009, and the *num iterations* and *early stopping rounds* that were set to the same values used in LGBM-Final (Table II). The model performance obtained on the training and validation datasets is shown with dashed and solid lines, respectively, while the red dashed lines in the ROC curves represent the performance of random classifiers. A summary of the performance obtained by the models shown in this figure can be visualized in Table VI.

Gradient boosting models											
Model	**Training**					**Validation**					**Training duration**
	F1-score	**MCC**	**TNR**	**TPR**	**Precision**	**F1**	**MCC**	**TNR**	**TPR**	**Precision**	
LGBM-1	0.7226	0.4524	0.8064	0.6382	0.7475	0.7216	0.4502	0.8055	0.6370	0.7463	0h:44m:46s
LGBM-Final	0.7518	0.5082	0.8189	0.6832	0.7722	0.7312	0.4665	0.7977	0.6635	0.7466	1h:24m:14s
Neural network models											
Model	**Training**					**Validation**					**Training duration**
	F1-score	**MCC**	**TNR**	**TPR**	**Precision**	**F1**	**MCC**	**TNR**	**TPR**	**Precision**	
NN-1	0.7265	0.4568	0.7914	0.6605	0.7399	0.7239	0.4513	0.7872	0.6594	0.7357	4h:57m:06s
CNN-1	0.7306	0.4646	0.7929	0.6671	0.7431	0.7299	0.4638	0.7950	0.6637	0.7442	3h:48m:55s
CNN-2	0.7347	0.4741	0.8048	0.6634	0.7532	0.7342	0.4733	0.8052	0.6620	0.7533	7h:41m:21s
CNN-Final	0.7434	0.4927	0.8189	0.6666	0.7678	0.7410	0.4879	0.8165	0.6643	0.7648	8h:43m:52s

TABLE VI

Summary of performance for model selection. The table shows the training time and a few accuracy measures obtained on the training and validation datasets from some of the models that we implemented in this study to perform model selection. The hyperparameters of the tuned gradient boosting model are shown in Table II, while the hyperparameters and the architectures of the neural networks are shown in Tables III and IV. LGBM-1 represents the baseline gradient boosting model, and its hyperparameters are set to default values except for the *learning rate*, which was set to 0.009, and the *num iterations* and *early stopping rounds* that were set as shown in Table II. The accuracy measures shown in the table are F1-score (macro average), Matthews correlation coefficient (MCC), true negative rate (TNR), true positive rate (TPR or recall), and precision. As explained in Materials and methods, each model can be considered an ensemble because the final prediction is obtained by averaging the probabilities predicted by the seven models obtained during the 7-folds CV. Also, the training duration refers to the total time used to perform the seven CV iterations that we used to train and validate each model.

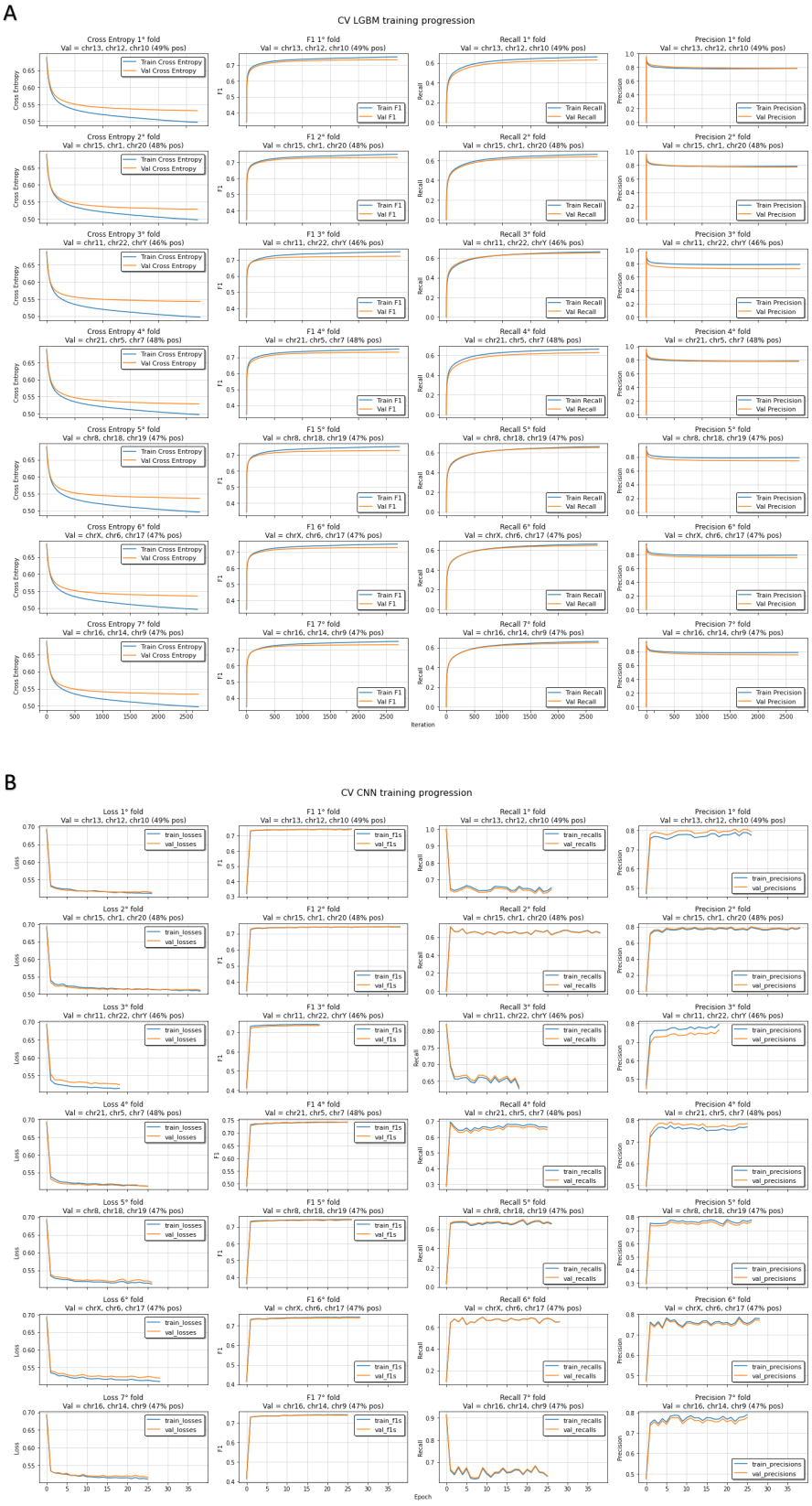

Fig. 9. **Training progression.** The figure shows the training progression of the final gradient boosting (**A**) and deep learning (**B**) models. Each row shows the progressive change of the model performance in one of the seven CV iterations used for training and validation, while each column shows a different metric: cross-entropy loss, F1-score, recall, and precision. The *x-axis* shows the number of training cycles (epochs for CNN or boosting iterations for LGBM), while the y-axis shows the value of the relative metrics obtained at a particular training cycle. The figure also shows the chromosomes used as validation and the percentage of positive samples in the validation set at each CV iteration. As explained in Materials and methods, the model performance of the CNN was obtained using a custom Keras callback [27]. This allowed us to estimate the real training and validation performance starting from models with untrained weights and biases (epoch 0). For visualization purposes, early stopping was not active on the LGBM training process because, since the model was evaluated using different metrics, early stopping would have arrested the training as soon as one of the evaluated metrics would have stopped improving.

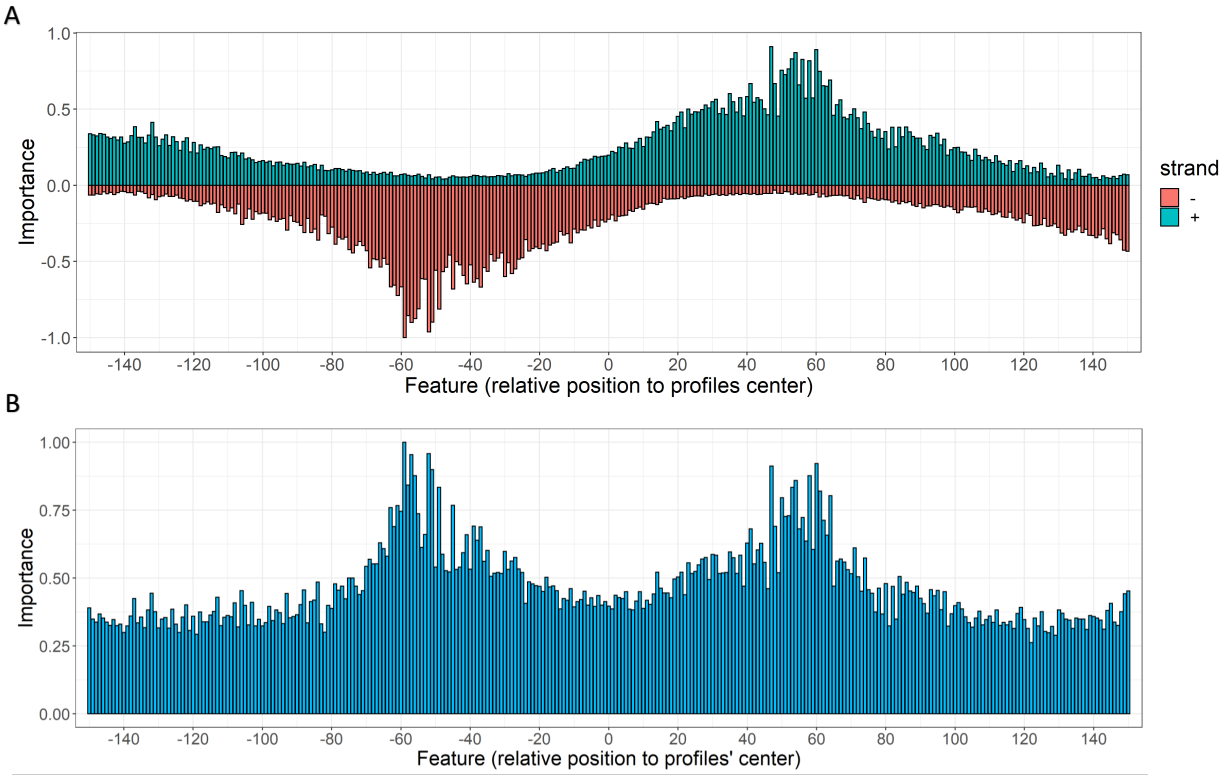

Fig. 10. **Feature importance.** The figure shows the feature importance, defined as the average information gain obtained by each feature during the training process, computed by the gradient boosting algorithm. **A** displays the feature importance obtained from the concatenated profiles, showing the importance of having CTSS at specific profile' positions on each strand. **B** shows the feature importance of the processed profiles. Consequently, it shows the importance of the difference in the CAGE signal intensity between strands at each position. For visualization purposes, in **A**, we assigned negative values to the importance of the reverse strand positions.

Feature vector	LGBM final model					CNN final model				
	F1-score	MCC	TNR	TPR	Precision	F1-score	MCC	TNR	TPR	Precision
Trainining (GM, ATAC)	0.7518	0.5082	0.8189	0.6832	0.7722	0.7434	0.4927	0.8189	0.6666	0.7678
Validation (GM, ATAC)	0.7312	0.4665	0.7977	0.6635	0.7466	0.7410	0.4879	0.8165	0.6643	0.7648
Test-1 (GM, ATAC)	0.7448	0.4948	0.8135	0.6755	0.7692	0.7511	0.5085	0.8249	0.6767	0.7804
Test-2 (GM, DNase)	0.7986	0.5981	0.7884	0.8106	0.7731	0.8053	0.6112	0.7974	0.8147	0.7815
Test-3 (HeLa, DNase)	0.7858	0.5748	0.7555	0.8202	0.7480	0.7906	0.5846	0.7597	0.8257	0.7525
Test-4 (HeLa Rrp40, DNase)	0.8121	0.6323	0.7566	0.8748	0.7608	0.8174	0.6419	0.7657	0.8758	0.7678

TABLE VII

Summary of model performance on training, validation, and test sets. The table shows the performance (the accuracy measures are described in Table VI) of the final gradient boosting (LGBM) and deep learning (CNN) models obtained on the different feature vectors generated in this study (Tables I and V). It is worth mentioning that the results shown in the table, as well as the other results shown in this study, are obtained using the optimal classification threshold (0.479 for LGBM, and 0.492 for CNN) computed by the threshold-moving technique, as explained in Materials and methods.

suggests that the models learned the critical patterns of the positive and negative profiles achieving good generalization ability. Slightly better predictions were obtained on *test-1*, which as the training (and validation) feature vector, was generated using ATAC-seq. Considerably better results were obtained on the feature vectors generated by the integration of CAGE and DNase-seq data from both GM12878 (*test-2*) and HeLa cell lines (*test-3*, and *test-4*), with both models obtaining the best performance on the test set obtained from HeLa cells with non-functional exosome complex (*test-4*). Looking at the confusion matrices of Figure 11, we can see that the performance improvement obtained on the datasets mentioned above was due to improved predictions of the positive samples (increased TPR), which came at the cost of an albeit minor decrease in performance obtained on the negative ones (decreased TNR), especially in the feature vectors generated from the HeLa cells' data (*test-3* and *test-4*). This is particularly evident in the results obtained on the feature vector generated from CAGE data sequenced from HeLa cells depleted of functional exosome (*test-4*), where both models obtained highly accurate predictions with $\approx$87% of true positive and $\approx$76% of true negative predicted samples.

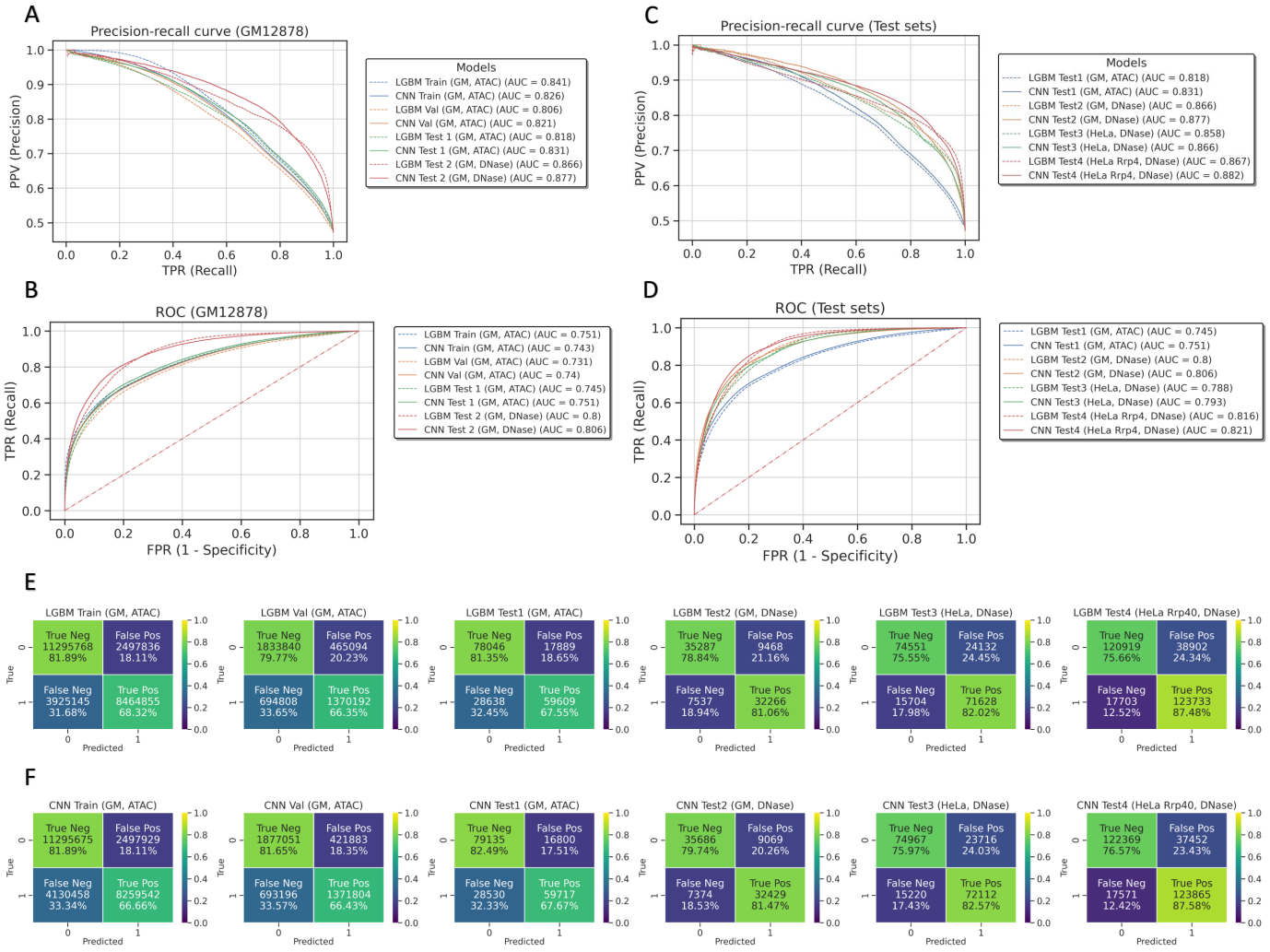

Fig. 11. **Model performance on training, validation, and test sets.** The figure shows the final CNN and LGBM models' performance on the different feature vectors generated in this study (Tables I and V). Plots **A** and **B** show the precision-recall and ROC curves computed on the predictions obtained on the feature vectors extracted from the GM12878 cell line, while **C** and **D** show the precision-recall and ROC curves evaluating the predictions achieved on the different test sets. As in Figure 8, the legends include the corresponding AUCs. Also, the gradient boosting model performance is shown with dashed lines, while solid ones display the performance obtained by the deep learning model. Plots **E** and **F** contain different heatmaps showing the confusion matrices computed on the results obtained by the gradient boosting and the deep learning model on each feature vector, including the true negative, true positive, false negative, false positive samples as percentages and counts.

To better understand the models' predictions, we analyzed the predicted average CAGE profiles and probabilities of the correctly and wrongly classified samples. Figure 12 shows the true and predicted average CAGE profiles of the positive and negative samples extracted from CAGE and DNase-seq data obtained from HeLa cells (*test-4*). As expected, we can observe that the average profiles of the true positive and true negative predicted samples are very similar to the ground truth. The main visible difference is that the average profile of the true negative predicted samples shows no CAGE signal on the forward strand downstream the profile' center and on the reverse strand upstream the profile' center, which is present in the true average profile of the negative samples. Instead, the average profile of the false negative predicted samples seems very noisy, with most of the signal located at its edges. In addition, also the average profile of the false positive predicted samples shows an abundance of noise. However, even if most of the CTSSs are located at its center, it remotely resembles the average profile of the true positive ones, suggesting a sign of bidirectional transcriptional initiation activity. In Figure 12 C and D, we can observe the distribution of the probabilities predicted by the models. In D, we can see that both models perform the majority of predictions with certainty, having the largest number of predicted probabilities on the extremes of the distribution. C shows the predicted probabilities that led to true negative, false positive, false negative, and true positive predictions. We can see that many true positive predicted samples were very easy to classify. In fact, the models had no doubt they were active transcriptional regulatory elements, assigning them probabilities larger than

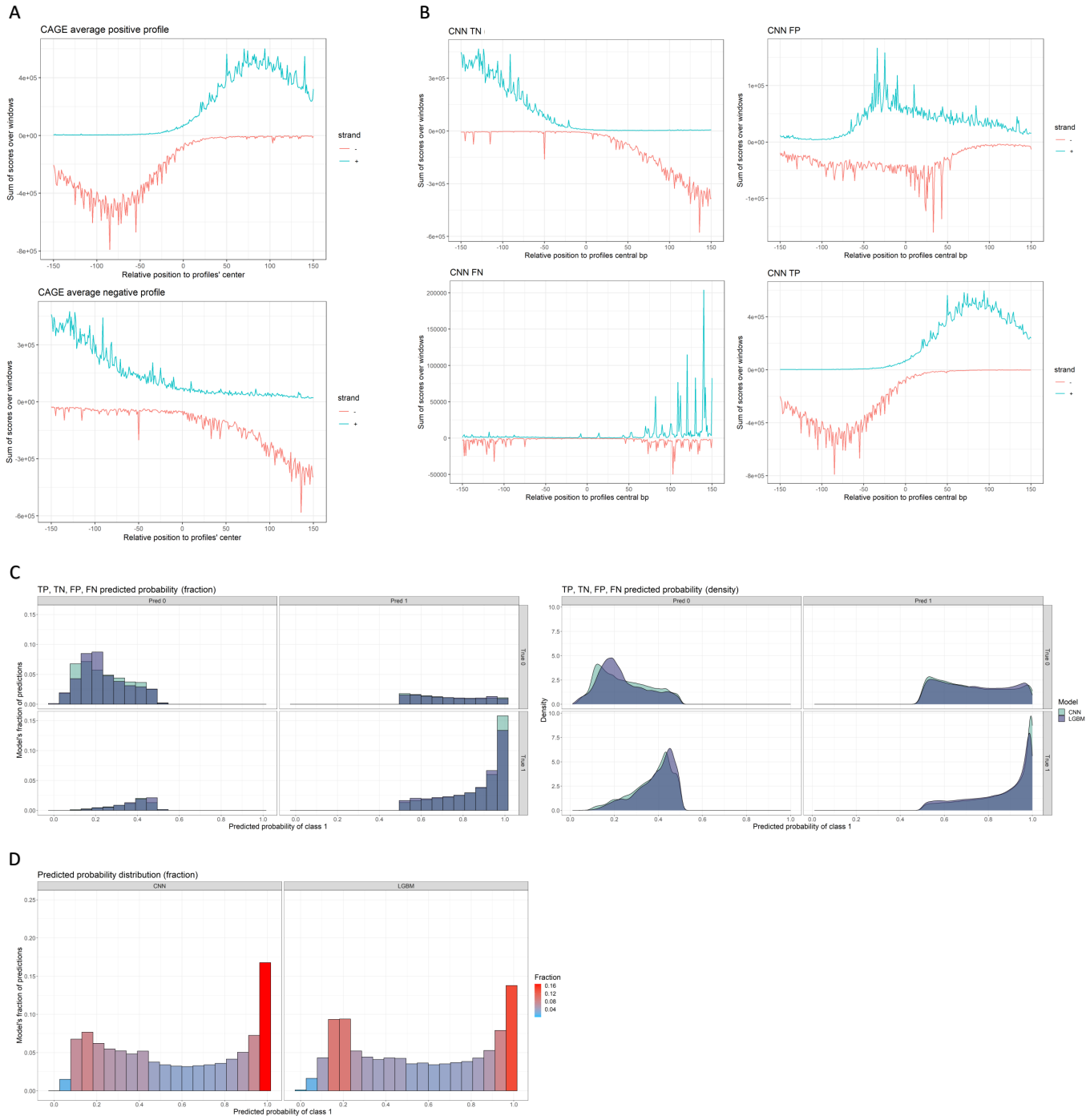

Fig. 12. **Predicted profiles and probabilities of samples from HeLa cell line.** The figure shows the average CAGE profiles of the true positive and negative samples (**A**) obtained from the *test-1* feature vector (samples obtained by the intersection of CAGE and DNase data from HeLa cell line), as well as the correctly and wrongly predicted profiles (**B**) and the distribution of the predicted probabilities (**C** and **D**). **B** displays the CAGE average profiles of the samples predicted by the CNN model. It includes the average profiles of the true negative (TN, top left), false positive (FP, top right), false negative (FN, bottom left), and true positive (TP, bottom right) predictions. **C** shows the distribution, both as a fraction (left) and as density (right), of the predicted probabilities of wrongly and correctly classified samples. It includes TN, FP, FN, and TP predictions performed by both CNN and LGBM. Lastly, **D** shows the distribution of all predicted probabilities (as a fraction of predictions) obtained by the two models. It is worth mentioning that the predicted probability refers to the probability that a particular profile is a positive sample, therefore an active transcriptional regulatory element.

0.9, while most of the true negative predicted samples were classified with probabilities ranging between 0.1 and 0.3. On the contrary, it is possible to observe that the models were not sure about the predictions that led to improperly classified profiles (false negative and false positive predictions), having most of the predicted probabilities ranging between 0.4 and 0.6. However, the models also made a certain number of wrong predictions, especially false positive ones, with certainty. This indicated that these predictions were performed on samples that were labeled as negative but showed a transcriptional initiation activity very similar to that of the positive ones. A similar pattern was found analyzing the predicted probabilities obtained on the training data (not shown in Results). We hypothesized that the samples that were wrongly predicted with certainty might have had a negative impact on the training process, confounding the learning of the models' parameters. Therefore, we filtered the training set by removing all false positive samples predicted with a probability larger than 0.7 and all false negative ones predicted with a probability less than 0.3. Then, we retrained the models on the training data without the potentially confounding profiles, and we evaluated their generalization performance on the test sets. However, unfortunately, this approach led to more unsatisfactory results (data not shown in Results).

We also generated two PCA plots (Figure 18 in Supplementary materials) showing the true positive and negative samples of *test-3*, as well as the correctly and wrongly predicted ones. However, most likely, the sparsity of the data did not allow the main principal components to capture enough variance for a meaningful interpretation of the plots.

Genome-wide prediction

The final goal of this study was to implement a command-line tool able to perform a genome-wide prediction of transcriptional regulatory elements from CAGE data. We developed a pipeline that takes as input the CTSSs data, scans the genome with a sliding window of 301 bp, and extracts the CAGE profiles of the actively transcribed regions having at least two CTSSs in one of their positions. We tested the tool, using both models and a step size of five bp, on one replicate of the in-house CAGE data obtained from HeLa cells. Scanning a window every five bp allowed the selection of 600,022,928 regions to be processed for the profile extraction. The tool required approximately three days to perform the extraction of the CAGE profiles, which is the most computationally

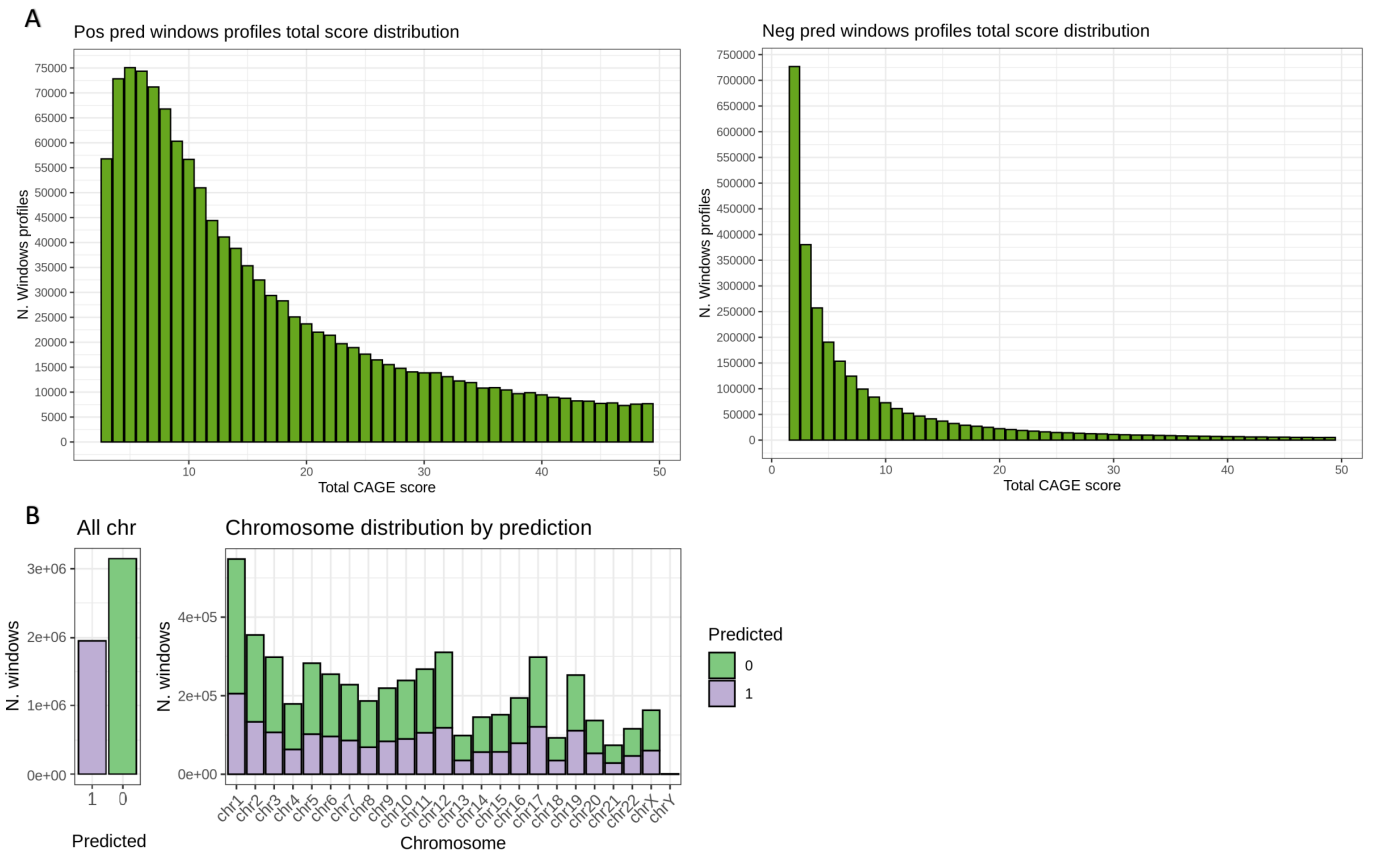

Fig. 13. **Genome-wide predictions (CTSS scores and sample distribution).** The figure shows some of the diagnostic plots produced by our tool, having as input one replicate (CTSS file) of the HeLa cell line. **A** shows the CTSS total scores distribution of the positive (left) and negative (right) samples predicted performing a genome-wide scan of transcriptionally active genomic regions (step size of 5 bp). The *y-axis* shows the count of predicted profiles having a particular total CTSS score, and the *x-axis* shows the total score obtained as a sum of CTSSs over individual profile' positions. For visualization purposes, we limited the coordinate of the *x-axis* showing only the number of profiles with a total CTSS score lower than 50. **B** shows the total number of positive and negative predicted samples (left) and the positive and negative predicted samples distribution over chromosomes (right).

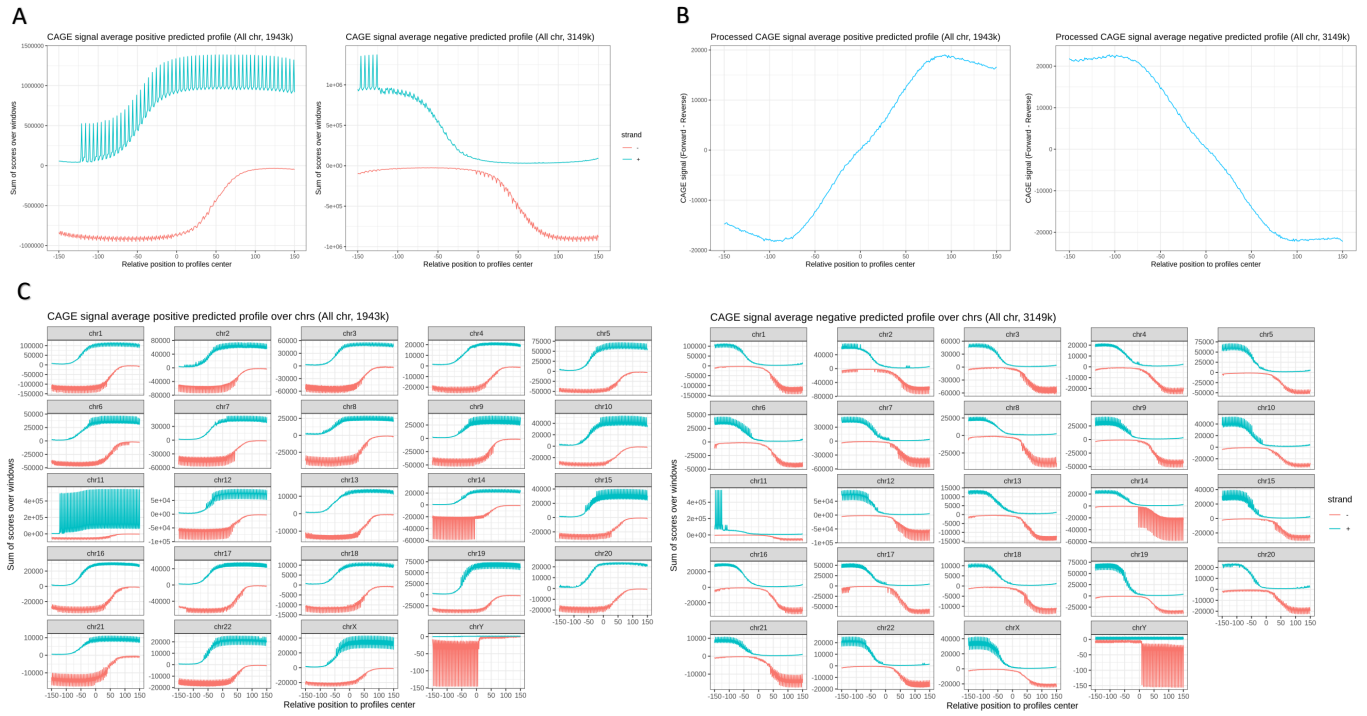

Fig. 14. **Genome-wide predictions (average CAGE profiles).** As Figure 13, this figure shows some of the diagnostic plots produced by our tool, such as the average CAGE profiles of the positive and negative genome-wide predicted samples. The prediction was performed on one replicate (CTSS file) from the HeLa cell line by scanning active genomic regions with a step of five bp, and using the gradient boosting model. **A** and **B** show the concatenated and processed average profiles, respectively, of positive (left) and negative (right) predicted samples, while **C** shows the concatenated profiles of positive (left) and negative (right) predicted samples at chromosomes resolution.

expensive process, and it obtained 20,032,083 regions with at least one TSS each. Next, it filtered the regions by the minimum requirement of transcription, obtaining 5,091,361 actively transcribed profiles. Then, it processed the profiles and used the trained models to perform the predictions, which required only a few minutes.

Figures 13 and 14 show some of the diagnostic plots produced by the tool as final output (gradient boosting predictions). Of the five million actively transcribed profiles extracted, 38.2% were predicted as positives, and 61.8% as negatives. In Figure 13, we can see the distribution of the predicted samples on the chromosomes, showing a similar ratio between positive and negative predicted profiles across all chromosomes. Figure 13 also shows the distribution of the total transcription initiation activity of the two classes predicted across the genome. As we observed in the average profiles of the samples labeled using chromatin accessibility (Figure 7), we can see (Figure 13) that the positive predicted regions are more transcribed than the negative predicted ones. In fact, for example, the largest number of profiles predicted as active TREs have a total of five CTSSs, while, as expected, the largest number of negative predicted regions show only the minimum level of transcriptional activity. Figure 14 shows the average CAGE profiles of the positive and negative predicted regions. In plots A and B, we can observe that the profiles of the predicted samples closely resemble those of the labeled ones generated using chromatin accessibility. In fact, the average profile of the active TREs candidates shows an S-shaped curve, suggesting bidirectional transcription initiation. In contrast, the profile of the regions predicted as negative seems to be a mirror image of the positive predicted ones, suggesting that most of these regions are flanking active OCRs, but their center is outside the open chromatin cores. Lastly, Figure 14 C shows the CAGE profiles of the predicted positive and negative samples at chromosome resolution, and, except for the spikes in the CAGE signal of some chromosomes, most of them closely resemble the average profiles observed in Figure 14 A. Interestingly, the average profiles obtained from chromosome 11 clearly show that most of the CAGE signal of the negative samples was expressed by regions flanking the positive ones.

As mentioned in Materials and methods, the tool's output includes a BED file which can be loaded into the UCSC Genome Browser [47] for interactive visualization of the candidate active TREs. Figure 15 shows an example of two frames obtained from the interactive visualization of the predictions. It shows the distribution of the TREs score predicted by both models taking as input one CAGE replicate obtained from HeLa cells (A) and one CAGE replicate obtained from GM12878 cells (B). It also includes tracks showing the CAGE signal, the regions of chromatin accessibility obtained by expanding the DNase-seq summits (A) and ATAC-seq peaks (B), and gene models information obtained from Ensembl [45]. The TREs score corresponds to the probability predicted by

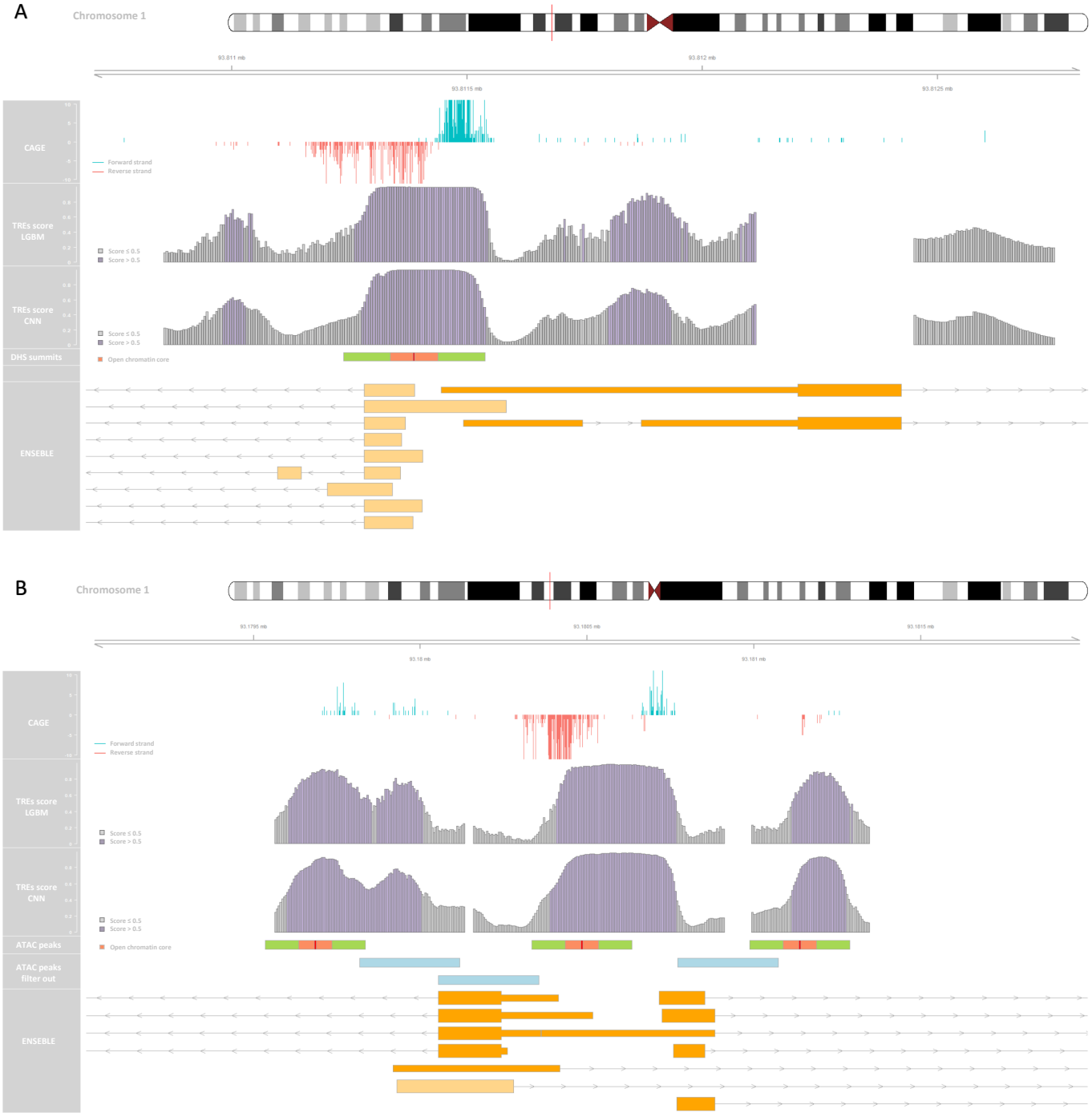

Fig. 15. **Genome-wide prediction (interactive visualization).** The figure shows two frames of an interactive visualization of the candidate active TREs predicted by our tool. The plots of this figure were generated using the Gviz [42] R package, but a similar visualization can also be performed by simply loading the BED file generated by our tool as a custom track on the UCSC Genome Browser [47]. The genome-wide prediction was performed on one CAGE replicate obtained from the HeLa cell line (**A**), and one CAGE replicate obtained from the timepoint 0 of the GM12878 (**B**) using both, gradient boosting and deep learning models. The two plots show two different genomic regions on chromosome 1 (the position on the chromosome is shown on the top of the figures). From top to bottom, the tracks display the CAGE signal on forward and reverse strands, the TREs scores predicted by LGBM and CNN, the extended DNase summits in **A** and ATAC peaks in **B**, the overlapping extended ATAC peaks filtered out by hierarchical approach in **B**, and the Ensembl [45] track showing gene models information. The TREs scores obtained by LGBM and CNN can be interpreted as the predicted probability that a particular genomic position is approximately at the center of an active transcriptional regulatory element. The loci having a predicted probability larger than 0.5 (classified as positive samples with default threshold) are displayed in light purple, while the ones with predicted probability equal or below 0.5 (classified as negative samples with default threshold) are shown in light gray. As in the CAGE average profiles displayed in the previous figures, in the CAGE track (first track on the top), we assigned negative values to the CAGE signal on the reverse strand. Also, for visualization purposes, we set a limit of +10 (forward strand) and -10 (reverse strand) on the coordinate system showing the CTSS scores.

the models that a particular genomic region of 301 bp is an active transcriptional regulatory element. The probability has been mapped to the central position of the selected region. Therefore, it can be interpreted as the probability that a specific genomic locus is approximately the center of an active open chromatin region. Observing both plots in Figure 15, we can see that the main difference between the gradient boosting and the CNN predictions is that the latter obtained a smoother distribution of the predicted scores, likely due to the spatial invariance provided by the convolutional and pooling layers. In Figure 15 A, we can observe that both models correctly predicted the displayed active OCR with a very high probability. However, even if with uncertainty, the models also predicted two regions lacking the open chromatin state as positive. In Figure 15 B, there are multiple overlapping ATAC-seq positive regions that we filtered using a hierarchical approach (Materials and methods). We can observe that the highest scores predicted by the models match the final filtered OCRs (green extended ATAC peaks), suggesting that the models obtained reasonable and meaningful results.

Discussion

In this study, we developed a user-friendly command line tool able to perform the genome-wide prediction of active transcriptional regulatory elements from TSS-focused sequencing data. The tool can accurately infer active TREs using a gradient boosting based on decision tree or a convolutional neural network model, depending on the user selection. Both models have been trained on a feature vector made of more than four million labeled samples generated by the integration of CAGE and ATAC-seq data.

The extraction of the feature vector was performed in such a way to allow the models to capture with flexibility the critical patterns of the transcriptional initiation activity observed in open chromatin regions. Also, it allowed the supervised machine learning algorithms to focus on learning the challenging transcriptional initiation profiles observed at the transition between active OCRs and proximal actively transcribed regions in closed chromatin states.

We carried out model selection by performing hyperparameters optimization and evaluating different neural network architectures. The best-performing CNN architecture (Tables III and VI) was obtained by combining three blocks of five stacked *convolutional layers* followed by a *max pooling layer*. In the last block, we used a *global max pooling layer*, followed by a *fully connected layer*, a *dropout layer*, and the final *output layer*. Although the model built with this architecture obtained the best out-of-sample performance among the neural networks, it is important to note that the simple baseline ANN already got reasonable results. Thus, suggesting that, from a modeling perspective, the prediction of active transcriptional regulatory elements might not be so difficult as to require highly complex and deep architectures.

By analyzing the importance of the positions of the profiles computed by the gradient boosting model, we observed that the most important predictors correspond to the positions of the positive profiles where the largest number of transcripts are initiated, which represent the distribution of core promoter TSS [17] in respect to the peak of highest chromatin accessibility of their associated NDRs.

To evaluate the reliability of the approach, we assessed the generalization performance of the models on four different test sets (Tables I and VII, and Figure 11), including samples obtained from two distinct cell lines (GM12878 and HeLa), and labeled using two alternative methods to determine chromatin accessibility (ATAC-seq and DNase-seq). Even if the convolutional neural network obtained the best results, both models showed very low signs of overfitting and excellent generalization performance. In addition, we observed that both models obtained more accurate predictions on the profiles labeled using DNase-seq, especially on the positive samples. The significant improvement in the TPR, observed on the predictions performed on these profiles, suggested that the samples labeled using ATAC-seq potentially included more wrongly assigned positive profiles than those labeled using DNase-seq. We hypothesized that this happened because the DHS index data produced by the Meuleman Lab from the integration of many different cell lines and tissues offered more robust and reliable sites of increased accessibility than the in-house ATAC-seq data. We speculated that another possible reason could be that Tn5 transposase might cut smaller DNA fragments than DNase-seq. Therefore, some peaks of increased accessibility identified by ATAC-seq could be short regions located between flanking nucleosomes that might not correspond to TREs. Moreover, the best model performance, with a remarkable improvement of the TPR, was achieved on the feature vector generated from the integration of DNase-seq and CAGE data obtained from HeLa cells deprecated of the nuclear exosome complex (*test-4*). This was entirely expected, in fact, it is known that the transcriptional regulatory elements show a bidirectional transcriptional initiation at the boundaries of nucleosome-depleted regions, but this is often not detected by steady-state RNA TSS sequencing techniques (e.g., CAGE) due to the degradation of short non-coding transcripts operated by the nuclear exosomes [17]. Therefore, the depletion of the exosome complex, which avoided the degradation of PROMPTs and eRNAs, allowed the models to better capture the patterns of transcriptional initiation observed at the TREs. This result suggested that our approach should achieve promising results on data generated by nascent-RNA TSS sequencing techniques such as GRO-cap, PRO-cap, and Start-seq.

To show the ultimate output of our tool, we ran it on one in-house CAGE replicate obtained from HeLa cells, performing the prediction of active TREs on five million actively transcribed regions extracted genome-wide using a step size of five bp (Figure 13). By analyzing the average CAGE profiles of the predicted windows, we observed (Figure 14) that the positive regions shown a characteristic S-shaped CAGE profile, suggesting the presence of bidirectional transcriptional initiation activity typical of TREs. In contrast, the average profile of the regions predicted as negative mostly resembles

a mirror image of the positive ones, suggesting that most of their signal is captured by regions flanking active OCRs. Moreover, we have shown (Figure 15) that our tool ultimately generates a distribution of predicted scores mapped to specific genomic positions, where each score represents the predicted probability that a particular locus is approximately the center of an active OCR. Thus, it allows the user to investigate different aspects of gene regulation, such as the genome-wide exploration of potential active transcriptional regulatory elements, from TSS profiling data alone.

One major limitation of our approach is that we modeled the regulatory elements as open chromatin regions of a fixed length, which is far from the reality of the diverse landscape of TREs. As proposed in the study performed by Danko *et al*. [31], a better approach could be to use a multiscale feature vector considering multiple window sizes, which, for example, might be optimized based on model performance.

Moreover, for a future perspective, we should investigate alternative evaluation strategies to objectively assess the method's reliability. In fact, we observed that the performance of the models could be biased by the reliability of the data used to define the ground truth, and more in general, by the effectiveness of the input extraction. A more unbiased evaluation strategy could involve comparing the enrichment of marks of transcriptional regulatory activities detected in candidate TREs predicted by our tool and by complementary methods.

Also, the current implementation of our tool does not allow the user to retrain the models. As prospects for the future, we would like to automate the labeled profiles extraction and the training process. This would enable the user to provide as input an indefinite number of CAGE and ATAC-seq or DNase-seq replicates that the tool would use to automatically extract a labeled feature vector and train and evaluate the model.

Additionally, it would be interesting to explore the feasibility of featuring the tool with the prediction of two additional scores, defined as the ability of a regulatory element to locally initiate transcription (promoter potential) and increase the rate of transcription in other regions (enhancer potential) [17], which will be mapped to the regions predicted as active TREs. This would represent a regression problem that will require the training of an additional model on a newly labeled feature vector. The two additional scores that could be used to label the predicted active TREs candidates could be determined, for example, by massively parallel reporter assays (MPRAs), which proved to be effective methods for determining regulatory elements' promoter and enhancer potential [17] [60]. Perhaps a simpler but not so effective approach could be to feature the tool with the ability to distinguish enhancers and promoters as two distinct classes. In this case, the new task would be a binary classification problem that could involve assigning labels of enhancer and promoter according to selected criteria. For example, the binary labels could be assigned based on the results obtained by MPRAs methods or based on the presence of specific patterns of epigenetic modifications, such as the combination of signals from H3K27ac, H3K4me1, and H3K4me3 histone modifications, which have been extensively used for the discrimination of gene promoters from enhancers [17]. Hypothetically, in either case (the assignment of promoter and enhancer potential or the simpler binary classification), the new target variables could be inferred by the model using as predictors the profiled transcriptional initiation activity, and the associated RNA abundance and sequence features, such as CpG dinucleotides content and the presence of particular motifs.

ACKNOWLEDGEMENT

I am extremely grateful to Robin Andersson and Marco Salvatore for allowing me to be part of this exciting study, providing guidance and supervision, and supporting my personal development as scientific researcher. I also want to express a special thanks to M. C. and my family, who always walk by my side.

SUPPLEMENTARY MATERIALS

Tool usage

Using the Linux command line [13], the user simply needs to specify the path of the CTSS input file using the flag `-i` or `--input_cage`, and the automated pipeline will generate the directories to store the results (if not already present), scan the genome extracting profiles of 301 bp, perform and store the predictions and generate the plots:

```
$ ./tool_master_script.sh -i <my_input_CTSS_file.bed.gz>
```

It is possible to add the flag `-h` or `--help` to get additional information about all available options:

```
$ ./tool_master_script.sh -h
```

It is recommended to specify an output filename using the flag `-f` or `--filename_out`, and an output directory using the flag `-o` or `--output_dir`:

```
$ ./tool_master_script.sh -i <my_input_CTSS_file.bed.gz>
  -f <my_output_filename> -o <my_output_directory/>
```

By default, the tool scans genomic regions every 5 bp, but the user can change the step size using the flag `-s` or `--step`:

```
$ ./tool_master_script.sh -i <my_input_CTSS_file.bed.gz>
  -f <my_output_filename> -o <my_output_directory/> -s <100>
```

By default, the format of the CTSS input file is GZ (compressed by Gzip [32]), but it can be changed to any format supported by the `rtracklayer::import()` function from the rtracklayer R package [51] [69] by using the flag `-F` or `--format`:

```
$ ./tool_master_script.sh -i <my_input_CTSS_file.bed>
  -f <my_output_filename> -o <my_output_directory/> -F <bed>
```

Supplementary figures

This subsection includes some figures supplementing the main text.

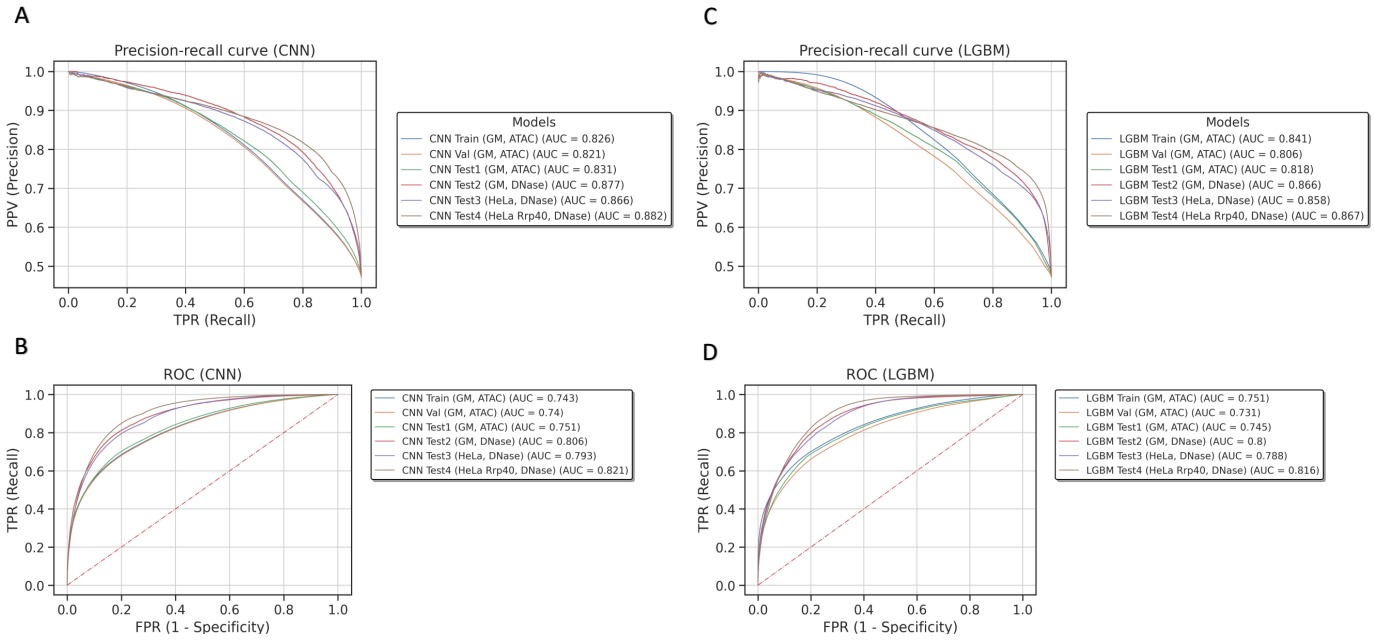

Fig. 16. **Model performance on training, validation, and test sets (focus on differences between feature vectors).** As in Figure 11, this figure shows the model performance on the feature vectors evaluated in this study. The main difference is that, while the precision-recall and ROC curves in Figure 11 focus to the comparison between LGBM and CNN models, the plots in this figure are arranged to show differences between the performance of the individual algorithms (CNN in plots **A** and **B**, while LGBM in **C** and **D**) on the different datasets.

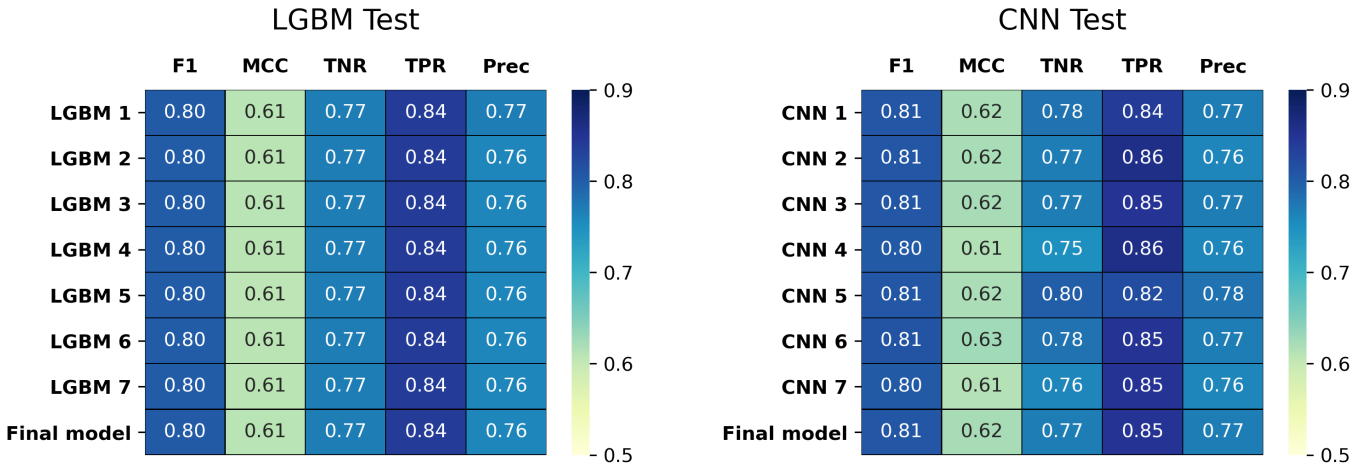

Fig. 17. **Prediction of new data.** The figure displays two heatmaps showing the performance obtained by the LGBM (left) and CNN (right) models on the feature vector generated from CAGE and DNase-seq data produced from GM12878 cells (*Test-2*). Each plot shows F1-score, MCC, TNR, TPR, and precision computed on the predictions performed by the seven models trained through the CV iterations. The last row shows the performance of the final ensemble, whose predictions are obtained by assigning labels according to the averaged probabilities predicted by the individual models mentioned above.

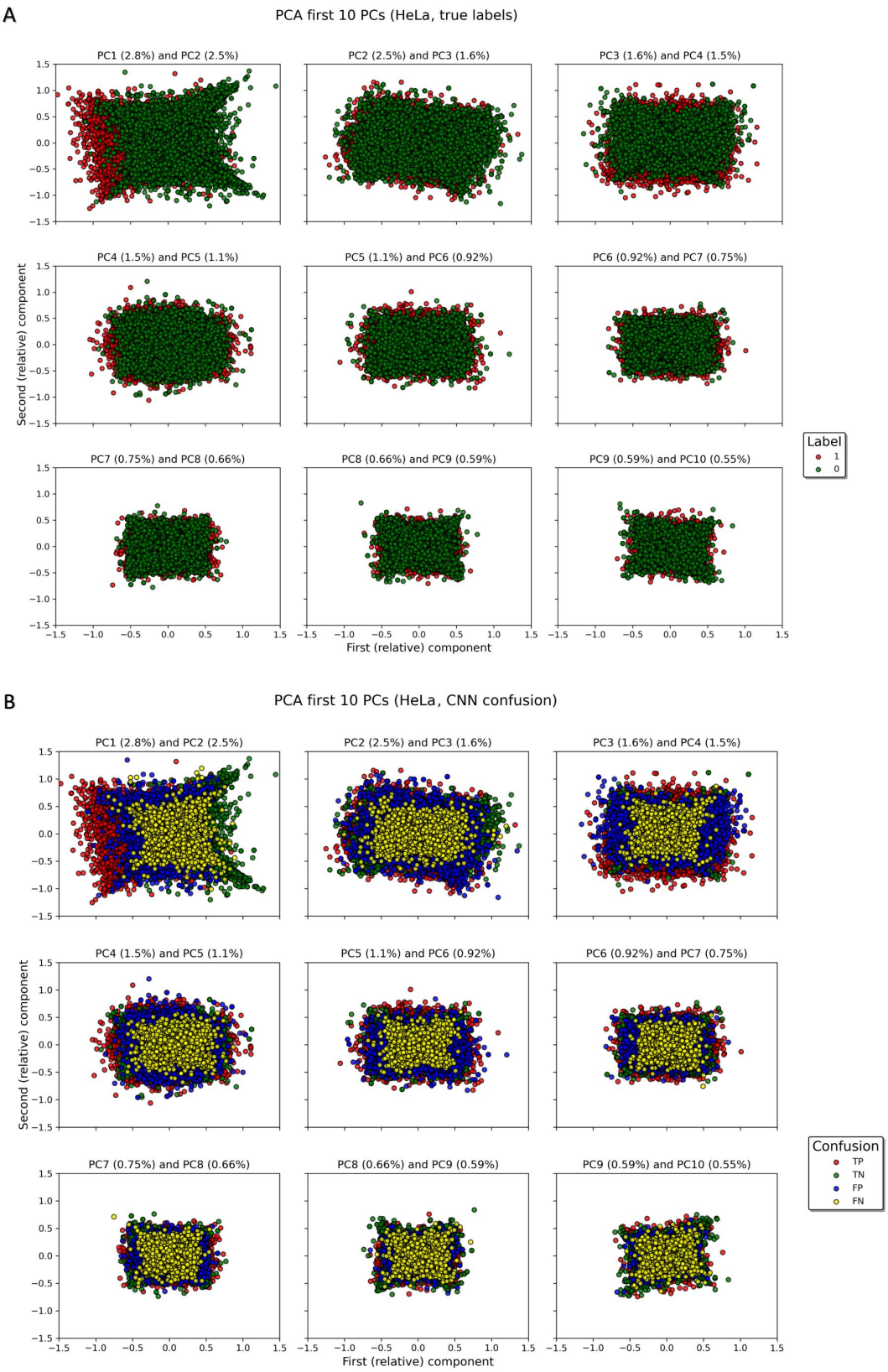

Fig. 18. **PCA plots of true and predicted samples from HeLa cell line. A** shows the PCA plot of the positive and negative samples (*test-3*), while **B** shows the correctly and wrongly classified ones, including the true positive , true negative, false positive, and false negative predictions. Each subplot of both plots shows the concatenated profiles of the samples projected on the two principal components indicated in the subtitle. The *x-axis* represents the first relative component, while the *y-axis* represents the second relative one. Unfortunately, the variance captured by each component, which is displayed in the subtitle of each subplot, is not enough to allow a meaningful interpretation.

References

[1] Andersson Lab, Bioinformatics Centre, University of Copenhagen.

[2] BayesianOptimization package Github. https://github.com/fmfn/BayesianOptimization.

[3] Encode project common cell types, National Human Genome Research Institute. https://www.genome.gov/encode-project-common-cell-types.

[4] Genome reference consortium. Human build 37. 2009. https://www.ncbi.nlm.nih.gov/assembly/GCF_000001405.13/.

[5] Genome reference consortium. Human build 38 patch release 13. 2019. https://www.ncbi.nlm.nih.gov/assembly/GCF_000001405.39#/def.

[6] Laurae blog: gradient boosting parameters. https://sites.google.com/view/lauraepp/parameters.

[7] Laurae Github. https://github.com/Laurae2.

[8] LightGBM documentation. https://eli5.readthedocs.io/en/latest/libraries/lightgbm.html.

[9] LightGBM documentation: parameters. https://lightgbm.readthedocs.io/en/latest/Parameters.html.

[10] LightGBM suggested parameters. https://github.com/Microsoft/LightGBM/issues/695.

[11] S. Pellegrini. Bioinformatics project, Andersson Lab, University of Copenhagen. Machine learning to infer active open chromatin regions from transcription initiation events. 2021.

[12] Wouter Meuleman. Annotated regulatory index. 2017-2020. https://www.meuleman.org/research/dhsindex/.

[13] The Linux Kernel Organization. https://www.kernel.org/code-of-conduct.html, 2021.

[14] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[15] H. M. Amemiya, A. Kundaje, and A. P. Boyle. The encode blacklist: identification of problematic regions of the genome. *Scientific reports*, 9(1):1–5, 2019.

[16] R. Andersson, P. R. Andersen, E. Valen, L. J. Core, J. Bornholdt, M. Boyd, T. H. Jensen, and A. Sandelin. Nuclear stability and transcriptional directionality separate functionally distinct rna species. *Nature communications*, 5(1):1–10, 2014.

[17] R. Andersson and A. Sandelin. Determinants of enhancer and promoter activities of regulatory elements. *Nature Reviews Genetics*, 21(2):71–87, 2020.

[18] P. Baldi and P. Sadowski. The dropout learning algorithm. *Artificial intelligence*, 210:78–122, 2014.

[19] S. D. Bolboacă. Medical diagnostic tests: a review of test anatomy, phases, and statistical treatment of data. *Computational and mathematical methods in medicine*, 2019, 2019.

[20] J. Brownlee. Blog: A gentle introduction to threshold-moving for imbalanced classification. https://machinelearningmastery.com/threshold-moving-for-imbalanced-classification/, 2020.

[21] J. Brownlee. Tensorflow tutorial: Classification on imbalanced data. https://www.tensorflow.org/tutorials/structured_data/imbalanced_data, 2020.

[22] J. D. Buenrostro, P. G. Giresi, L. C. Zaba, H. Y. Chang, and W. J. Greenleaf. Transposition of native chromatin for multimodal regulatory analysis and personal epigenomics. *Nature methods*, 10(12):1213, 2013.

[23] J. D. Buenrostro, B. Wu, H. Y. Chang, and W. J. Greenleaf. Atac-seq: a method for assaying chromatin accessibility genome-wide. *Current protocols in molecular biology*, 109(1):21–29, 2015.

[24] C. Cao, F. Liu, H. Tan, D. Song, W. Shu, W. Li, Y. Zhou, X. Bo, and Z. Xie. Deep learning and its applications in biomedicine. *Genomics, proteomics & bioinformatics*, 16(1):17–32, 2018.

[25] P. Carninci, A. Sandelin, B. Lenhard, S. Katayama, K. Shimokawa, J. Ponjavic, C. A. Semple, M. S. Taylor, P. G. Engström, M. C. Frith, et al. Genome-wide analysis of mammalian promoter architecture and evolution. *Nature genetics*, 38(6):626–635, 2006.

[26] S. Chatterjee and N. Ahituv. Gene regulatory elements, major drivers of human disease. *Annual review of genomics and human genetics*, 18:45–63, 2017.

[27] F. Chollet et al. Keras documentation: Callbacks API. https://keras.io/api/callbacks/.

[28] F. Chollet et al. Keras documentation: general questions. https://keras.io/getting_started/faq/.

[29] F. Chollet et al. Keras. https://keras.io, 2015.

[30] L. J. Core, A. L. Martins, C. G. Danko, C. T. Waters, A. Siepel, and J. T. Lis. Analysis of nascent rna identifies a unified architecture of initiation regions at mammalian promoters and enhancers. *Nature genetics*, 46(12):1311–1320, 2014.

[31] C. G. Danko, S. L. Hyland, L. J. Core, A. L. Martins, C. T. Waters, H. W. Lee, V. G. Cheung, W. L. Kraus, J. T. Lis, and A. Siepel. Identification of active transcriptional regulatory elements from gro-seq data. *Nature methods*, 12(5):433–438, 2015.

[32] P. Deutsch. Rfc1952: Gzip file format specification version 4.3, 1996.

[33] Q. Dong and G. Luo. Progress indication for deep learning model training: A feasibility demonstration. *IEEE Access*, 8:79811–79843, 2020.

[34] E. R. Dorman, A. M. Bushey, and V. G. Corces. The role of insulator elements in large-scale chromatin structure in interphase. In *Seminars in cell & developmental biology*, volume 18, pages 682–690. Elsevier, 2007.

[35] D. J. Epstein. Cis-regulatory mutations in human disease. *Briefings in Functional Genomics and Proteomics*, 8(4):310–316, 2009.

[36] C. M. Florkowski. Sensitivity, specificity, receiver-operating characteristic (roc) curves and likelihood ratios: communicating the performance of diagnostic tests. *The Clinical Biochemist Reviews*, 29(Suppl 1):S83, 2008.

[37] P. I. Frazier. A tutorial on bayesian optimization. *arXiv preprint arXiv:1807.02811*, 2018.

[38] D. Godoy. Blog: Understanding binary cross-entropy/log loss: a visual explanation, 2018.

[39] A. Gordon, G. Hannon, et al. Fastx-toolkit. http://hannonlab.cshl.edu/fastx_toolkit.

[40] E. Guresen and G. Kayakutlu. Definition of artificial neural networks with comparison to other networks. *Procedia Computer Science*, 3:426–433, 2011.

[41] V. Haberle and A. Stark. Eukaryotic core promoters and the functional basis of transcription initiation. *Nature reviews Molecular cell biology*, 19(10):621–637, 2018.

[42] F. Hahne and R. Ivanek. *Statistical Genomics: Methods and Protocols*, chapter Visualizing Genomic Data Using Gviz and Bioconductor, pages 335–351. Springer New York, New York, NY, 2016.

[43] S.-H. Han, K. W. Kim, S. Kim, and Y. C. Youn. Artificial neural network: understanding the basic concepts without mathematics. *Dementia and neurocognitive disorders*, 17(3):83, 2018.

[44] V. Hinman and G. Cary. Multicellularity: The evolution of gene regulation. *Elife*, 6:e27291, 2017.

[45] K. L. Howe, P. Achuthan, J. Allen, and Allen. Ensembl 2021. *Nucleic Acids Research*, 49(D1):D884–D891, 11 2020.

[46] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T. Liu. Lightgbm: A highly efficient gradient boosting decision tree. In *NIPS*, 2017.

[47] W. J. Kent, C. W. Sugnet, T. S. Furey, K. M. Roskin, T. H. Pringle, A. M. Zahler, and D. Haussler. The human genome browser at ucsc. *Genome research*, 12(6):996–1006, 2002.

[48] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[49] S. Kiranyaz, T. Ince, O. Abdeljaber, O. Avci, and M. Gabbouj. 1-d convolutional neural networks for signal processing applications. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 8360–8364. IEEE, 2019.

[50] H. Koohy. The rise and fall of machine learning methods in biomedical research. *F1000Research*, 6, 2017.

[51] M. Lawrence, R. Gentleman, and V. Carey. rtracklayer: an r package for interfacing with genome browsers. *Bioinformatics*, 25:1841–1842, 2009.

[52] H. Li and R. Durbin. Fast and accurate short read alignment with burrows–wheeler transform. *bioinformatics*, 25(14):1754–1760, 2009.

[53] Z. Li, M. H. Schulz, T. Look, M. Begemann, M. Zenke, and I. G. Costa. Identification of transcription factor binding sites using atac-seq. *Genome biology*, 20(1):1–21, 2019.

[54] M. W. Libbrecht and W. S. Noble. Machine learning applications in genetics and genomics. *Nature Reviews Genetics*, 16(6):321–332, 2015.

[55] G. A. Maston, S. K. Evans, and M. R. Green. Transcriptional regulatory elements in the human genome. *Annu. Rev. Genomics Hum. Genet.*, 7:29–59, 2006.

[56] W. Meuleman, A. Muratov, E. Rynes, J. Halow, K. Lee, D. Bates, M. Diegel, D. Dunn, F. Neri, A. Teodosiadis, et al. Index and biological spectrum of human dnase i hypersensitive sites. *Nature*, 584(7820):244–251, 2020.

[57] T. Mitsis, A. Efthimiadou, F. Bacopoulou, D. Vlachakis, G. P. Chrousos, and E. Eliopoulos. Transcription factors and evolution: an integral part of gene expression. *World Academy of Sciences Journal*, 2(1):3–8, 2020.

[58] M. Murat Arat. Blog: Cross-entropy for tensorflow. https://mmuratarat.github.io/about/, 2018.

[59] A. Natekin and A. Knoll. Gradient boosting machines, a tutorial. *Frontiers in neurorobotics*, 7:21, 2013.

[60] T. A. Nguyen, R. D. Jones, A. R. Snavely, A. R. Pfenning, R. Kirchner, M. Hemberg, and J. M. Gray. High-throughput functional comparison of promoter and enhancer activities. *Genome research*, 26(8):1023–1033, 2016.

[61] J. A. Nichols, H. W. H. Chan, and M. A. Baker. Machine learning: applications of artificial intelligence to imaging and diagnosis. *Biophysical reviews*, 11(1):111–118, 2019.

[62] R. Nott. Hela cell line. *Embryo Project Encyclopedia*, 2020.

[63] I. Ohn and Y. Kim. Smooth function approximation by deep neural networks with general activation functions. *Entropy*, 21(7):627, 2019.

[64] A. Panigrahi and B. W. O'Malley. Mechanisms of enhancer action: the known and the unknown. *Genome biology*, 22(1):1–30, 2021.

[65] K. H. Park, E. Batbaatar, Y. Piao, N. Theera-Umpon, and K. H. Ryu. Deep learning feature extraction approach for hematopoietic cancer subtype classification. *International Journal of Environmental Research and Public Health*, 18(4):2197, 2021.

[66] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[67] M. Pérez-Enciso and L. M. Zingaretti. A guide on deep learning for complex trait genomic prediction. *Genes*, 10(7):553, 2019.

[68] A. R. Quinlan and I. M. Hall. Bedtools: a flexible suite of utilities for comparing genomic features. *Bioinformatics*, 26(6):841–842, 2010.

[69] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2017.

[70] A. Rajkomar, J. Dean, and I. Kohane. Machine learning in medicine. *New England Journal of Medicine*, 380(14):1347–1358, 2019.

[71] M. Saqib, Y. Zhu, M. Wang, and B. Beaulieu-Jones. Regularization of deep neural networks for eeg seizure detection to mitigate overfitting. In *2020 IEEE 44th Annual Computers, Software, and Applications Conference (COMPSAC)*, pages 664–673. IEEE, 2020.

[72] A. N. Schep, J. D. Buenrostro, S. K. Denny, K. Schwartz, G. Sherlock, and W. J. Greenleaf. Structured nucleosome fingerprints enable high-resolution mapping of chromatin architecture within regulatory regions. *Genome research*, 25(11):1757–1770, 2015.

[73] H.-C. Shin, H. R. Roth, M. Gao, L. Lu, Z. Xu, I. Nogues, J. Yao, D. Mollura, and R. M. Summers. Deep convolutional neural networks for computer-aided detection: Cnn architectures, dataset characteristics and transfer learning. *IEEE transactions on medical imaging*, 35(5):1285–1298, 2016.

[74] T. Shiraki, S. Kondo, S. Katayama, K. Waki, T. Kasukawa, H. Kawaji, R. Kodzius, A. Watahiki, M. Nakamura, T. Arakawa, et al. Cap analysis gene expression for high-throughput analysis of transcriptional starting point and identification of promoter usage. *Proceedings of the National Academy of Sciences*, 100(26):15776–15781, 2003.

[75] A.-M. Šimundić. Measures of diagnostic accuracy: basic definitions. *Ejifcc*, 19(4):203, 2009.

[76] J. Snoek, H. Larochelle, and R. P. Adams. Practical bayesian optimization of machine learning algorithms. *arXiv preprint arXiv:1206.2944*, 2012.

[77] L. Song and G. E. Crawford. Dnase-seq: a high-resolution technique for mapping active gene regulatory elements across the genome from mammalian cells. *Cold Spring Harbor Protocols*, 2010(2):pdb–prot5384, 2010.

[78] Y.-Y. Song and L. Ying. Decision tree methods: applications for classification and prediction. *Shanghai archives of psychiatry*, 27(2):130, 2015.

[79] I. Tabian, H. Fu, and Z. Sharif Khodaei. A convolutional neural network for impact detection and characterization of complex composite structures. *Sensors*, 19(22):4933, 2019.

[80] A. Tahmassebi, A. H. Gandomi, S. Fong, A. Meyer-Baese, and S. Y. Foo. Multi-stage optimization of a deep model: A case study on ground motion modeling. *PloS one*, 13(9):e0203829, 2018.

[81] H. Takahashi, S. Kato, M. Murata, and P. Carninci. Cage (cap analysis of gene expression): a protocol for the detection of promoter and transcriptional networks. In *Gene regulatory networks*, pages 181–200. Springer, 2012.

[82] M. Thodberg, A. Thieffry, K. Vitting-Seerup, R. Andersson, and A. Sandelin. Cagefightr: analysis of 5-end data using r/bioconductor. *BMC bioinformatics*, 20(1):1–13, 2019.

[83] T. Toh, F. Dondelinger, and D. Wang. Looking beyond the hype: Applied ai and machine learning in translational medicine. ebiomedicine, 47, 607–615, 2019.

[84] G. Tseng. Gradient boosting and XGBoost. https://medium.com/@gabrieltseng/gradient-boosting-and-xgboost-c306c1bcfaf5, 2018.

[85] G. Van Rossum and F. L. Drake Jr. *Python reference manual*. Centrum voor Wiskunde en Informatica Amsterdam, 1995.

[86] Z. Wang, T. Chu, L. A. Choate, and C. G. Danko. Identification of regulatory elements from nascent transcription using dreg. *Genome research*, 29(2):293–303, 2019.

[87] P. J. Wittkopp and G. Kalay. Cis-regulatory elements: molecular mechanisms and evolutionary processes underlying divergence. *Nature Reviews Genetics*, 13(1):59–69, 2012.

[88] R. Yamashita, M. Nishio, R. K. G. Do, and K. Togashi. Convolutional neural networks: an overview and application in radiology. *Insights into imaging*, 9(4):611–629, 2018.

[89] Y. Zhang, T. Liu, C. A. Meyer, J. Eeckhoute, D. S. Johnson, B. E. Bernstein, C. Nusbaum, R. M. Myers, M. Brown, W. Li, et al. Model-based analysis of chip-seq (macs). *Genome biology*, 9(9):1–9, 2008.

[90] Z. Zhang. A gentle introduction to artificial neural networks. *Annals of translational medicine*, 4(19), 2016.

[91] J. Zhong, K. Luo, P. S. Winter, G. E. Crawford, E. S. Iversen, and A. J. Hartemink. Mapping nucleosome positions using dnase-seq. *Genome research*, 26(3):351–364, 2016.

[92] Z.-H. Zhou. *Ensemble methods: foundations and algorithms*. CRC press, 2012.