Machine learning to infer active open chromatin regions from transcription initiation events

Stefano Pellegrini
Faculty of SCIENCE
University of Copenhagen
(MLQ211)

Fundamental components of gene regulation, such as transcriptional regulatory elements (TREs), can be revealed by cap analysis of gene expression (CAGE), a highly accurate 5'-end sequencing technique that can be used to profile transcription initiation activity. In this pilot study, we investigate the feasibility of alternative approaches for the identification of potential active TREs from transcription start site (TSS) -focused sequencing data.

We generated a multiscale feature vector from CAGE data, which was fed into three different machine learning algorithms. Input exploratory analysis showed evidence of the meaningfulness of the input extraction strategy, while performance evaluation indicated a gradient boosting framework based on decision tree (GBDT) as our best method and confirmed the suitability of the approach.

Based on our results we are confident that this study can form the basis for a future implementation of a solid framework for the prediction of potential active TREs using TSS-focused sequencing data.

INTRODUCTION

Understanding the underlying mechanism of gene regulation is one of the most important challenges in biomedical research [21], and transcriptional regulatory elements (TREs), including both gene promoters and enhancers, are key components of the genetic regulatory processes [8]. Established high-throughput genomic approaches for the identification of TREs, such as chromatin immunoprecipitation and synthesis (ChIP-Seq) and DNase I hypersensitivity and sequencing (DNase-Seq), do not provide direct evidence of regulatory function, and they often detect elements that are not actively transcribed [5] [10]. In 2015, Danko et al. [10] proposed a method for the detection of actively transcribed TREs from global run-on sequencing (GRO-Seq) or precision run-on sequencing (PRO-Seq) data. The key idea of their study was that active TREs are located in regions with accessible chromatin and, they present characteristic patterns of divergent transcription [6] [5] that can be detected by machine learning algorithms [10]. An alternative method for revealing patterns of divergent transcription is the quantification of transcription initiation activity. Such activity can be measured using cap analysis of gene expression (CAGE), a highly accurate sequencing technique capturing the 5'-end of RNA transcripts. However, from a machine learning perspective, the possibility of using transcription start site (TSS) -based sequencing data for the prediction of active TREs remains unexplored.

In this pilot project, we investigate the feasibility of alternative approaches that use the relationship between divergent, bidirectional transcription initiation and open chromatin to detect potential active regulatory elements. Our method uses CAGE data to profile transcription initiation activity, assay for transposase-accessible chromatin using sequencing (ATAC-Seq) data to detect open chromatin regions, and supervised machine learning algorithms to identify the characteristic patterns of transcription at TREs. Eventually, this study will provide the foundation for a future implementation of a robust machine learning architecture for the prediction and exploration of active TREs.

MATERIALS AND METHODS

Data

The models developed in this pilot project are built using in-house CAGE and ATAC-Seq data obtained from GM12878 lymphoblastoid cell line [2] cultured in Andersson Lab [1]. Trimming and mapping of the CAGE data was obtained respectively by FASTX-Toolkit [13] and Burrows-Wheeler aligner [16]. We used BEDTools [19] to pool the CAGE tag start site (CTSS) signals from four replicates into a single browser extensible data (BED) file, containing transcription initiation scores for 4142444 genomic positions. The ATAC-Seq data included 258265 significant peaks, defining sites of open chromatin regions, which were obtained by MACS2 peak caller algorithm [25].

Input extraction

We faced the problem of building a framework for the prediction of potential active regulatory regions as a binary classification problem. The basic idea behind supervised classification methods is to use labeled data to train predictive models. To fulfill this task, we got inspiration from Danko's study [10] and we generated a multiscale feature vector from the CAGE scores. Also, we defined specific criteria for assigning positive and negative labels to the samples.

The positive set (potential active TREs) was defined such that it included actively transcribed and nucleosome-free nonoverlapping genomic regions. That was possible by intersecting the CAGE signal with the ATAC-Seq positive peaks. We started by extending the ATAC peaks of 250 bp in both

directions, generating 258265 open chromatin regions of 501 bp. We removed the regions overlapping with the ENCODE blacklist [4] and then we extracted the CAGE profile of the resulting windows. Each profile corresponded to a single vector of 1002 elements, containing the CAGE scores mapped to the 501 window positions on both forward and reverse strands. The extraction generated 138277 transcribed window profiles. These included multiple intra-overlapping regions that were removed by keeping only windows with the highest rank, resulting in 87407 positive samples. The rank was computed as the sum of the CAGE scores across the windows positions. These profiles were filtered by a CAGE signal requirement, selecting only windows containing at least a TSS with a score of 2, yielding the final positive set of size 60522.

The negative set was defined in such a way to include regions of matching CAGE signal requirement but lacking marks of chromatin accessibility. It was generated by extracting 4417984 nonoverlapping genomic ranges of 501 bp. The regions were extracted randomly from the *hg38* genome using BEDTools [19]. The task was performed in such a way that the negative windows did not overlap: any open chromatin region identified by ATAC-Seq, each other, and any range included in the ENCODE blacklist [4]. Subsequently, we extracted the profiles of these windows, identifying 411257 transcribed regions. Alternative filtering approaches were tested to generate different negative sets. Ultimately, we chose to use the same filter adopted for the positive set extraction, keeping only the profiles showing at least a TSS with a score of 2, yielding 159550 windows. Sampling without replacement was performed to obtain two subsets of the extracted negative profiles: one with the same amount of profiles contained in the positive set (60522), and the other of halved size (30261).

Each negative set was merged with the positive one, generating three datasets mainly differing by the sample size ratio between negative and positive sets (1, 0.5, and $\sim$2.5). Each resulting dataset was shuffled and split into train and test sets, assigning samples from chromosomes 2, 3, and 4 to the test data, and all other samples to the training data.

In addition to BEDTools [19], the generation of the multi-scale feature vector was obtained using R/Bioconductor [20] [12] and GenomicRanges R-package [15]. Also, we used ggplot2 R-package [24] to generate plots for the assessment of the quality and the meaningfulness of the resulting profiles.

Models development

We trained and evaluated three different models for the prediction of potential active TREs: a random forest, a gradient boosting framework based on decision tree algorithm (GBDT), and a support vector machine (SVM).

Both random forest and GBDT methods are ensemble learning algorithms that consist of a large number of decision trees working as a committee [22] [14]. A decision tree is a flowchart-like structure where each internal node corresponds to a test that splits the data based on a cutoff for a certain attribute. However, in random forest, each decision tree is trained on an independent dataset which is obtained by random sampling with replacement (bootstrapping) from the original training data, and the model final predictions are obtained by equally weighting their votes [7] [22]. Differently, boosting algorithms are based on the idea that it is possible to build a good ensemble model by starting from a weak classifier and recursively adding a new weak classifier that is based on the mistakes made by the previous ones [26]. In fact, at each new iteration, the algorithm uses gradient descent to fit a new decision tree on the residual errors, which are computed as the difference between the true values and the weighted sum of the previous models' predictions [17]. Distinctly, SVM is a very different algorithm that aims to find the optimal decision boundary that separates the different classes. For a non-linear classification problem, SVM uses the kernel trick to minimize the classification error and maximize the soft margin [9]. Specifically, we used the radial basis function (RBF) kernel to correctly divide the classes in such a way that that the distance between the decision boundary and the closest data points (the margin) is maximized [9]. Simply speaking, the kernel trick applies a non-linear transformation to existing features to generate new features that are used to find a new non-linear decision boundary structure [9]. The term *soft* in the soft margin indicates that a certain degree of misclassification is tolerated [9].

Random forest and SVM were implemented using Scikit-learn [18] machine learning library, while GBDT was implemented using LightGBM [14]. In general, the code necessary to build, run, and evaluate the models is written in Python [23].

To perform validation and hyperparameters tuning, we implemented a cross validation (CV) function that splits the data into train and test according to chromosomes. By default, the number of chromosomes used for validation is set to three and the function generates the required number of folds to validate the models against all chromosomes. At each CV iteration, the function normalizes the data to zero mean and one unit variance by applying Scikit-learn [18] *standard scaler* (fitted on the training data and applied on both train and validation), it performs training and prediction of the relative fold, it evaluates the model performance reporting the accuracy, F1-score, Matthews correlation coefficient (MCC), true positive rate (TPR), true negative rate (TNR) and, lastly, it stores the model feature importance (in random forest and LightGBM). When all iterations are computed, the function outputs the final model performance across all folds, the predicted labels and their probabilities, the true labels, and the feature importance averaged across the folds. To perform hyperparameters optimization, we implemented a grid and random search function using Scikit-learn [18], but, due to the models' time complexity and the limited time we had for this study, we focused only on few parameters, which are shown in Table 1. Particular care was taken to tune and apply class weights (this parameter is reported only for LightGBM) to balance the classification results obtained on the three datasets differing by negative and positive sample size.

Random Forest			
Parameter	**Value**	**Parameter**	**Value**
N. estimators	100	Criterion	Gini
Bootstrap	True	Max features	Auto
Min sample leaf	1	Min sample split	2
LightGBM			
Parameter	**Value**	**Parameter**	**Value**
N. iterations	1500	Application	Binary
Learning rate	0.009	Metric	Binary logloss
Scale pos. weight	1.65	Device	GPU
GPU platform ID	0	GPU device ID	0
SVM			
Parameter	**Value**	**Parameter**	**Value**
Kernel	RBF	C	100
Probability	True	Gamma	0.1

TABLE I
MODELS HYPERPARAMETERS. *The table shows the hyperparameters of random forest, LightGBM, and SVM. All other hyperparameters are set to default setting.*

As mentioned, during CV we monitored each model's performance but we also stored their predictions and the actual labels. This data was used to generate plots evaluating and comparing the performance of the models obtained on both training and validation data. Also, to estimate the models' generalization ability, we compared these results to the ones obtained by the models used to predict the final test set, showing both in-sample and out-of-sample accuracy measures.

RESULTS

We used CAGE and ATAC-Seq data to generate a multiscale feature vector, consisting of the transcription initiation profiles of ∼1-kbp nonoverlapping genomic windows. The feature vector was then used to train and evaluate three supervised machine learning models (random forest, LightGBM, and SVM) finalized at the prediction of potential active TREs. To train the models, we assigned positive labels to the profiles of potential active regulatory regions revealed by the intersection between CAGE and ATAC-Seq signals, and negative labels to the profile of regions matching the CAGE signal requirement but lacking evidence of nucleosome depletion.

Input exploratory analysis

The meaningfulness of the extracted profiles was assessed by analyzing the shapes of their CAGE signal. Fig 1 shows the CAGE cumulative score distribution across the positions of the windows profiles, for both positive and negative samples. Recalling that these positions are defined in relation to the ATAC-Seq peak, it is interesting to note that the CAGE signal of the positive set shows a sigmoidal shape centered at position 0, indicating bidirectional and divergent transcription initiation activity [6]. On the positive set, we can see that most CAGE scores are located between positions -175 and 25 on the forward strand, and between positions -25 and 175 on the reverse strand. Contrarily, the CAGE scores of the negative samples show an overall uniform distribution over the windows positions, with some peaks particularly present on the reverse strand. Observing the CAGE score distribution with chromosomes resolution (Fig 2), we can see an overall similar pattern. However, since the axes are not fixed to a certain coordinate range, some chromosomes in both sets present extremely high peaks that hide the actual shape of the signal. Anyhow, we can see that chromosome Y has a very low signal, especially in the positive set where it only shows a single transcription start site (TSS) with a score of 2.

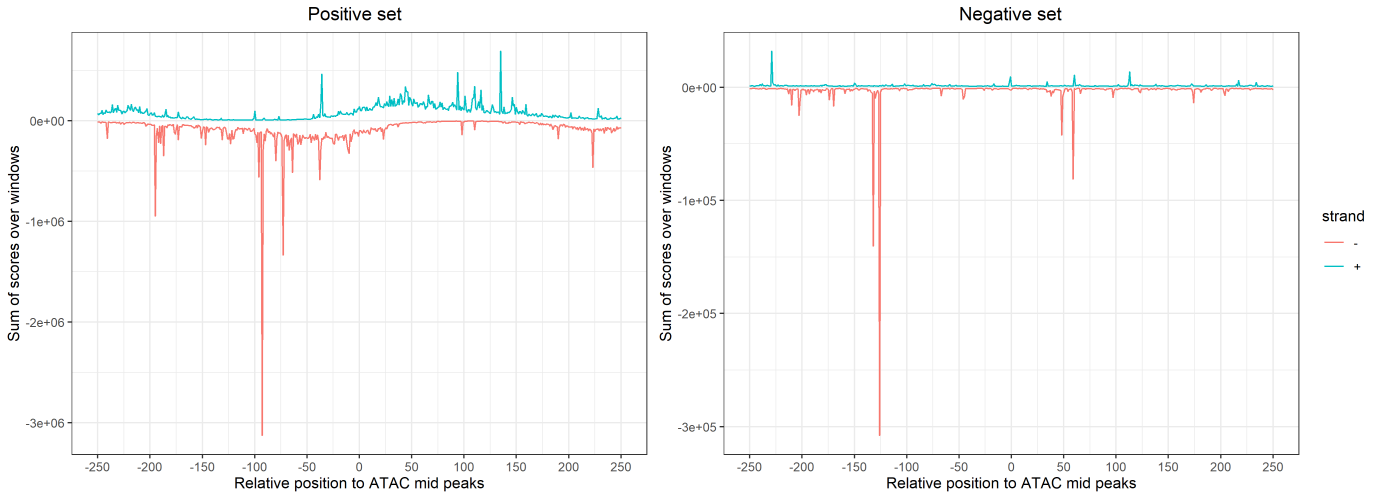

Fig. 1. **CAGE scores distribution.** The figure shows the cumulative distribution of the CAGE scores over the ATAC-Seq peaks relative position, for both positive (on the left) and negative samples (on the right). The *x* axis represents the relative position in respect to the ATAC-Seq peak, while the *y* axis indicates the cumulative sum of the CAGE scores at a certain window position over all genomic windows. For visualization purposes, we assigned a negative value to the CAGE signal on the reverse strand.

Fig. 2. **CAGE scores distribution over chromosomes.** Similarly to Fig. 1, but at chromosomes resolution, the figure shows the cumulative distribution of the CAGE scores over the ATAC-Seq peaks relative positions.

Models evaluation

We trained and evaluated three classification algorithms on three different datasets, mainly differing by the ratio between the number of negative and positive samples: 1, 0.5, and ∼2.5. Initially, we started evaluating the balanced dataset (ratio of 1), and we observed that the models predicted the negative samples better than the positive ones. Therefore, we halved the negative set (ratio of 0.5), which increased the accuracy obtained on the positive samples but resulted in overall poorer performance. Due to the strategy used to generate the input, we had a smaller availability of positive samples than negative ones. Therefore, since the quantity of data can often limit the performance of a machine learning model, we used all available negative samples (ratio of ∼2.5) and we added class weights to deal with the unbalanced classes. However, this strategy did not improve the models' performance, so we decided to rely on the dataset with an equal number of positive and negative samples, which is the one used to generate all plots shown in this section.

To provide insights into the model and the data, we started by plotting the feature importance of random forest and LightGBM, which is shown in Fig 3. As expected, the two algorithms showed very similar but not identical results. In fact, the Scikit-learn [18] implementation of random forest, based on Breiman's paper [7], calculates the feature importance as a measure of the information gain computed in respect to the Gini impurity objective function [3]. While for LGBM [14], we choose to measure the importance as the average gain of a certain feature when it is used in the trees. It is possible to observe that the feature importance distribution is very similar to the sigmoidal shape of the positive set CAGE signal (Fig. 1), with the most important features ranging from position -100 to 20 on the reverse strand and from position -20 to 100 on the forward strand. This suggests that a correct extraction of the positive set is essential to obtain predictive models with reasonable learning ability. Also, it might indicate that the models learn more useful information from the positive samples than the negative ones, which could be the reason why increasing the negative set size did not lead to better performance.

As previously stated, we tuned and applied class weights to balance the classification results, meaning that we let the models penalize mistakes in samples of one class more or less than the other. This was useful for dealing with the unbalanced datasets, but it was particularly useful for training the SVM and, especially, the LightGBM, on the balanced data. In fact, while random forest obtained a reasonable balanced classification performance, these models obtained much better results on the negative samples than the positive ones. However, while applying class weights to the SVM did not drastically improve the predictions of the positive samples, applying them to LightGBM led to reasonably good results. This can be observed in Fig. 4, which shows the progressive change of the loss function and other metrics, during the LightGBM training process in one CV iteration.

Fig. 5 shows the F1-score and MCC obtained by the models

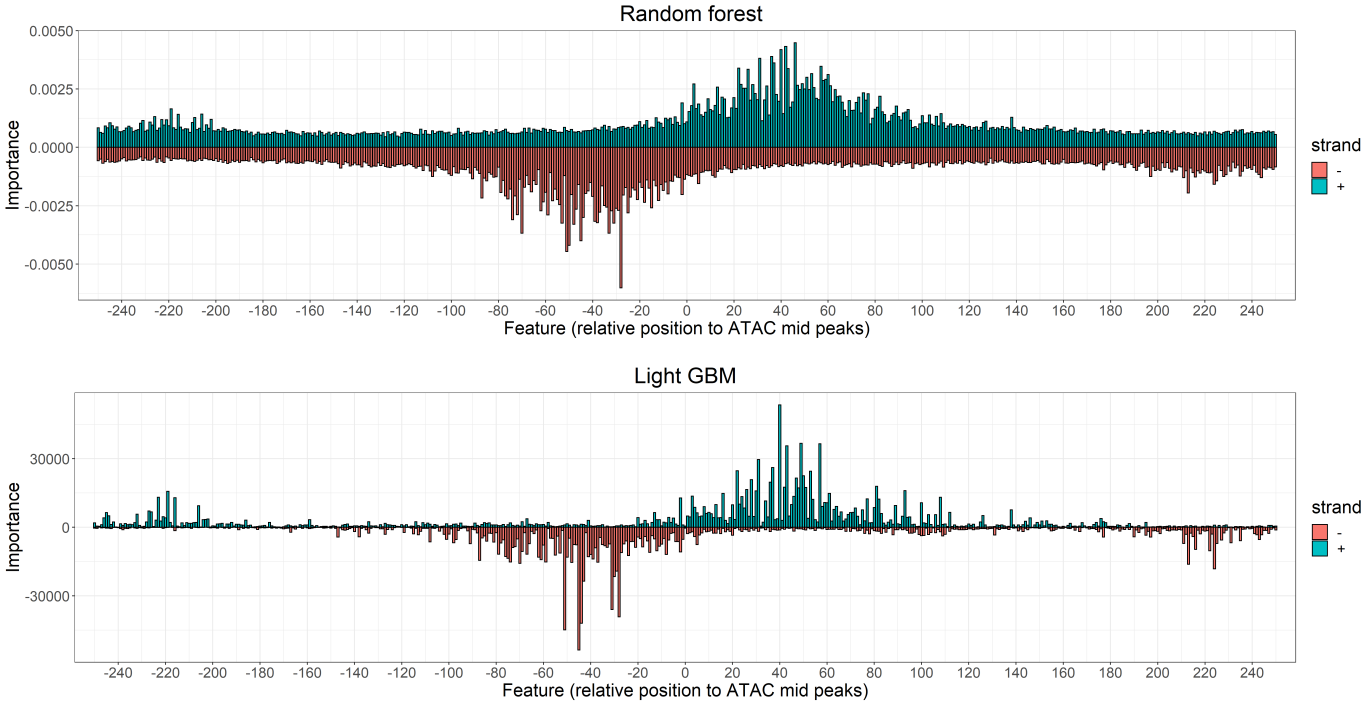

Fig. 3. **Feature importance.** The figure shows the feature importance of the random forest (top) and LightGBM (bottom) models. The scores are obtained by averaging the importance of the features through all cross validation iterations. The *x* axis represents the models' features, which correspond to the genomic windows positions defined in relation to the ATAC-Seq peaks. The *y* axis shows the importance of each feature, computed by each classification algorithm. As in Fig. 1 and 2, a negative value was assigned to the importance of the reverse strand positions.

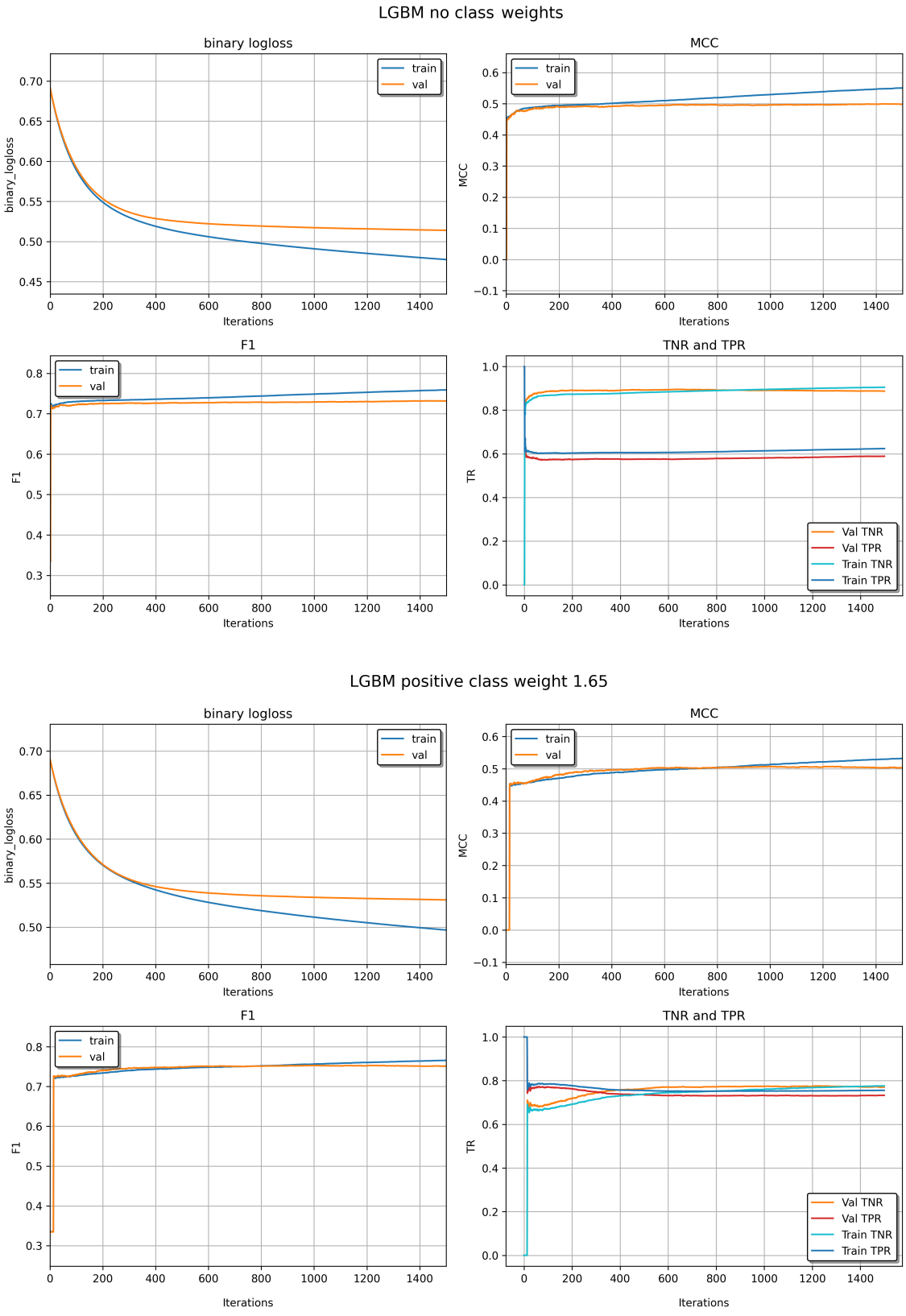

Fig. 4. **LightGBM training process with and without class weights.** The figure shows the LightGBM training process evaluation during one CV iteration using the model with (bottom) and without (top) weighting the positive samples error (weight of 1.65). In this example, the model is trained on all chromosomes included in the training data, except chromosome 10, 12, and 13, which are used as validation. The metrics, shown for both training and validation data, are binary log loss (subplot top left), MCC (subplot top right), F1-score (subplot bottom left), TNR and TPR (subplot bottom right).

during CV (*Val CV*) and the prediction of the final test set (*Test*). The figure shows both in-sample (warm colors) and out-of-sample (cold colors) performance. As mentioned, the results are obtained on the balanced dataset, therefore, the F1-scores equal the models' accuracy. We can see that LightGBM was the best performing algorithm, showing highest F1-score and MCC, almost no sign of overfitting, and very robust learning ability. Contrarily, random forest clearly overfitted the training data, while SVM showed an inconspicuous sign of overfitting but obtained the worst out-of-sample performance on both validation and test sets.

In Fig. 6, we can see the performance of the models at classes resolution, including the confusion matrices obtained from the predictions performed on the training and validation/test data, and a summary plot showing TNR and TPR. It is possible to observe that, as previously stated, SVM obtained the most unbalanced classification performance, showing a poor predictive power on the positive samples and a very good performance on the negative ones. Since we tuned the added positive penalty, LightGBM obtained equally good TNR and TPR on the validation data, while, on the test data, it obtained better results on the negative samples than the positive ones. Also, as expected, TNR and TPR confirmed the random forest overfitting of the training data. Lastly, in both Fig. 5 and 6, we can see that all models show very similar performance between validation and test data, suggesting substantially similar patterns between the two datasets, and therefore, between chromosomes.

Fig. 7 is the last figure of this section. It includes a receiver operating characteristic (ROC) curve, its log version, and a precision-recall curve. The ROC and its log version show recall (TPR) versus specificity (TNR or 1 - FPR) for every possible classification threshold, while the precision-recall curve, as its name implies, shows the trade-off between recall and precision [11]. It is possible to observe that LightGBM was confirmed as the best model on both validation and test data, while SVM obtained the worst performance.

Table 2 includes a summary of the models' performance, also showing the time needed by the models to perform cross validation. We can see that LightGBM was the fastest model, completing all iterations in less than 10 minutes, random forest required 45 minutes, while SVM was the slowest one, requiring 15 hours to complete the CV.

Metric	Random forest	LightGBM	SVM
F1 (CV Train)	0.96	0.77	0.78
F1 (CV Val)	0.73	0.74	0.71
F1 (Test)	0.73	0.75	0.72
MCC (CV Train)	0.92	0.53	0.6
MCC (CV Val)	0.46	0.48	0.45
MCC (Test)	0.47	0.5	0.47
TNR (CV Train)	1	0.78	0.93
TNR (CV Val)	0.74	0.74	0.84
TNR (Test)	0.79	0.79	0.88
TPR (CV Train)	0.92	0.76	0.64
TPR (CV Val)	0.68	0.74	0.6
TPR (Test)	0.68	0.71	0.55
AUC (CV Train)	0.95	0.77	0.79
AUC (CV Val)	0.72	0.74	0.72
AUC (Test)	0.73	0.75	0.72
CV Duration	00h:45m:27s	00h:08m:58s	15h:00m:44s

TABLE II
SUMMARY OF THE MODELS' PERFORMANCE. *The table shows a summary of the models' performance. For clarity, we omitted the performance obtained on the complete training data which was instead included in most figures of this section.*

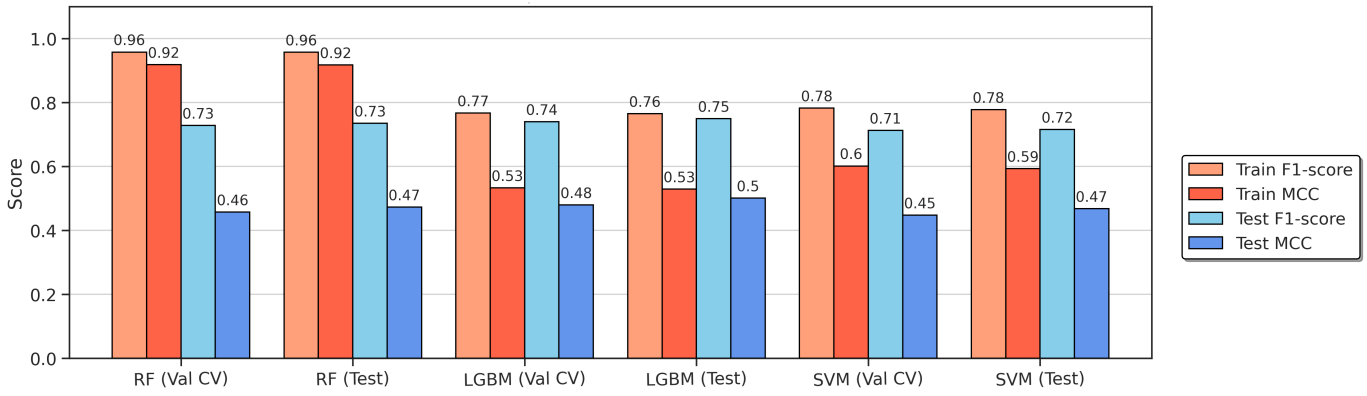

Fig. 5. **F1-score and MCC.** The figure shows the F1-score and MCC performances obtained by the three algorithms during cross validation (*Val CV*) and the test set prediction (*Test*). The bars of warm colors represent the performance of the models obtained on the training set during CV (a subset of the data) and on the training set used to train the final model (complete training data). The cold colors indicate the model performance on validation and test datasets. The exact scores of the models are reported on top of the bars.

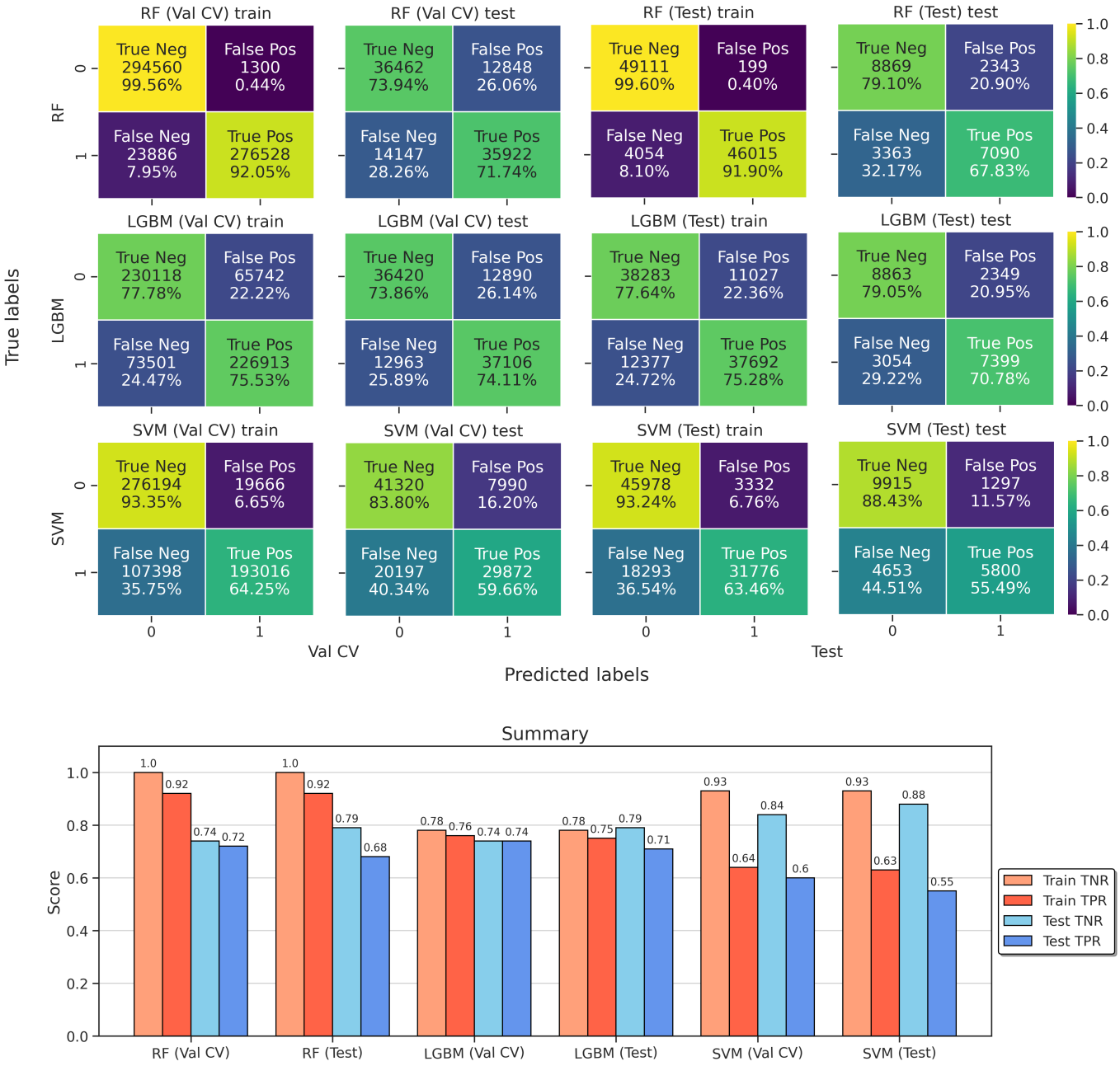

Fig. 6. **Confusion matrices and summary bar plot.** The figure (top) shows the confusion matrices computed on the predictions performed during CV (*Val CV*) and on the final test set (*Test*). A summary bar plot, showing TNR and TPR for the different classification algorithms, is shown at the bottom. In the confusion matrices, the x axis represents the models' predicted labels, while the y axis represents the true ones. The matrices are arranged such that each row of plots corresponds to a single classification algorithm, while the columns divide the performance obtained during CV (the two on the left) from the ones obtained during the prediction of the test set (the two on the right). The in-sample performance is shown in the first column (on a subset of training data used during CV) and the third one (on the complete training data used to train the final models), while the out-of-sample performance obtained on the validation and test sets is shown respectively in the second and fourth columns. Also, in each subplot is shown the number of falsely and correctly predicted classes and the corresponding percentage rate. In the summary bar plot, as in Fig. 5, the models' performance obtained on the train data are shown in warm colors, the ones obtained on the test and validation data are shown in cold colors, and the exact scores obtained by the models are reported on top of the bars.

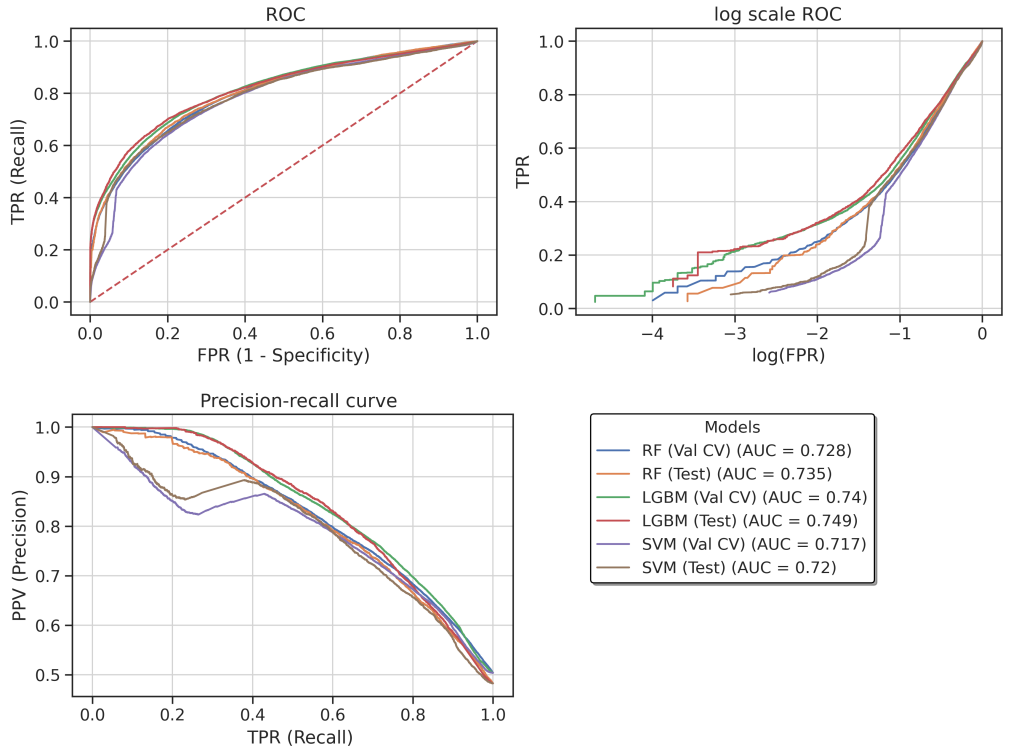

Fig. 7. **ROC and precision-recall curves.** The figure shows the ROC curve (top left), its log scale version (top right), and the precision-recall curve (bottom left) of the models performance obtained on the validation (CV) and test data. The dashed red line in the ROC curve (top left) indicates the performance of a random classifier, while the area under the curve (AUC) value of each model is shown in the legend. The x axis shows the FPR and its log value in the ROC curves, and the TPR in the precision-recall one. The y axis shows the TPR in the ROC curves and the positive predicted values (PPV or precision) in the precision-recall one.

DISCUSSION

In this study, we have explored the feasibility of the prediction of potential active regulatory regions from 5'-end sequencing data, obtaining promising results. We have seen that, even with some limitations, we generated a multiscale feature vector from CAGE and ATAC-Seq data, which was used to feed different supervised classification algorithms. We have observed that the CAGE signal of the extracted positive profiles presented the typical bidirectional and divergent transcription initiation activity of the active regulatory elements [6]. Contrarily, the CAGE scores of the negative set profiles were uniformly distributed on the windows positions. Furthermore, we have shown that the most important model features matched the most expressed sites of the positive profiles and that increasing the negative set sample size did not increase the models' performance. In this regard, we could speculate that, while the models were able to learn characteristic patterns from the CAGE positive signal, they did not capture useful information from the negative one, which might be mainly produced by noisy data. We have tested three different algorithms (random forest, LightGBM, and SVM) and we clearly identified LightGBM as the best one, showing very robust and fast learning ability, almost no sign of overfitting, and best overall performance. The gradient boosting method also showed a good capability to obtain balanced predictions of the two classes, even with unbalanced datasets.

In light of our results, we are confident that this pilot project can provide the foundation for a future implementation of a solid framework for the prediction of active TREs using TSS-focused sequencing data. As future remarks, it will be important to improve the quality and the quantity of the labeled data. For example, we could investigate alternative strategies for the extraction of the multiscale vector and use signal smoothing methods to remove noise from the data. Also, we could use data augmentation to increase both positive and negative sample size. Regarding model selection, we could start from LightGBM and perform parameter tuning by grid search or Bayesian optimization. Then, we could explore different methods, such as convolutional neural networks (CNN). Also, to increase the model's generalization performance, we will need to consider different CAGE datasets and, eventually, other TSS-based data such as GRO-cap and Start-Seq. Lastly, it will be important to investigate effective evaluation strategies. In fact, the results obtained in this study are strictly dependent on the effectiveness of the method used for the input extraction. To achieve a more objective evaluation, it will be crucial to perform an enrichment analysis of the TREs predicted by our method and compare the outcome with the results obtained by other methods.

ACKNOWLEDGEMENT

I am grateful to Robin Andersson and Marco Salvatore for giving me the chance to be part of this exciting project, for providing supervision and guidance, and, most importantly, for supporting and shaping my personal development as a scientific researcher.

REFERENCES

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

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

[3] Random forest classifier, Scikit-learn documentation. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.

[4] 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.

[5] R. Andersson, C. Gebhard, I. Miguel-Escalada, I. Hoof, J. Bornholdt, M. Boyd, Y. Chen, X. Zhao, C. Schmidl, T. Suzuki, et al. An atlas of active enhancers across human cell types and tissues. *Nature*, 507(7493):455–461, 2014.

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

[7] L. Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[8] E. Calo and J. Wysocka. Modification of enhancer chromatin: what, how, and why? *Molecular cell*, 49(5):825–837, 2013.

[9] C. Cortes and V. Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[10] 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.

[11] 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.

[12] R. C. Gentleman, V. J. Carey, D. M. Bates, B. Bolstad, M. Dettling, S. Dudoit, B. Ellis, L. Gautier, Y. Ge, J. Gentry, et al. Bioconductor: open software development for computational biology and bioinformatics. *Genome biology*, 5(10):R80, 2004.

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

[14] 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.

[15] M. Lawrence, W. Huber, H. Pagès, P. Aboyoun, M. Carlson, R. Gentleman, M. Morgan, and V. Carey. Software for computing and annotating genomic ranges. *PLoS Computational Biology*, 9, 2013.

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

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

[18] 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.

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

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

[21] A. Sandelin. Prediction of regulatory elements. In *Bioinformatics*, pages 233–244. Springer, 2008.

[22] A. Sarica, A. Cerasa, and A. Quattrone. Random forest algorithm for the classification of neuroimaging data in alzheimer's disease: A systematic review. *Frontiers in aging neuroscience*, 9:329, 2017.

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

[24] H. Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York, 2016.

[25] 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.

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