PREDICTION OF THE REGULATORY REGIONS ACTIVE IN HEK293 THROUGH DEEP LEARNING METHODS

Davide D'Ascenzo
University of Milan
davide.dascenzo@studenti.unimi.it

ABSTRACT

Neural networks are an increasingly popular solution for many machine learning problems. In this work, the task we want to tackle is related to the realm of Bioinformatics. In particular, we want to predict the activity of regulatory regions in the 'HEK293' cell line, using three different types of neural networks: multilayer perceptron, convolutional neural network and multi-modal neural network.

1 Introduction

Bioinformatics is an interdisciplinary field that combines biology, computer science, information engineering, mathematics and statistics to analyze and interpret the biological data [1]. It uses computer programs for a variety of applications, including determining gene and protein functions, establishing evolutionary relationships, and predicting the three-dimensional shapes of proteins [2].

In this work, we want to analyze regulatory regions in 'HEK293' immortalized cell line and determine which regions are active and which not.

1.1 Regulatory regions

A regulatory region is a region of the DNA that controls gene expression. Those regions are capable of increasing or decreasing the expression of specific genes within an organism. There are two types of regions: cis-regulatory regions (also called cis-regulatory elements) and trans-regulatory regions (also called trans-regulatory elements). We are interested in two particular cis-regulatory elements: 'promoters' and 'enhancers'.

1.1.1 Promoter

A promoter is a sequence of DNA to which proteins bind that initiate transcription of a single RNA from the nearby DNA. Promoters are located near the transcription start sites of genes and can be about 100–1000 base pairs (bp) long, the sequence of which is highly dependent on the gene and product of transcription, type or class of RNA polymerase recruited to the site and species of organism. [3]

1.1.2 Enhancer

An enhancer is a short (50–1500 bp) region of DNA that can be bound by proteins (activators) to increase the likelihood that transcription of a particular gene will occur [4][5]. These proteins are usually referred to as transcription factors.

1.2 Immortalized cell line

An immortalised cell line is a population of cells from a multicellular organism which would normally not proliferate indefinitely but, due to mutation, have evaded normal cellular senescence and instead can keep undergoing division. The cells can therefore be grown for prolonged periods in vitro. [6]

1.2.1 HEK293

Human embryonic kidney 293 cells, also often referred to as HEK 293, HEK-293, 293 cells, or less precisely as HEK cells, are a specific immortalised cell line originally derived from human embryonic kidney cells grown in tissue culture taken from a female fetus [7].

2 Models

In this work, deep neural network regressors are built to predict the activity of regulatory regions in 'HEK293'. We use three different types of neural network architectures: the multilayer perceptron, the convolutional neural network and the multi-modal neural network.

All the architectures illustrated in the following subsections are handcrafted, since model selection would take too much time in the environment used.

2.1 Deep neural network

A deep neural network (DNN) is an artificial neural network (ANN) with multiple layers between the input and output layers [8][9]. There are different types of neural networks but they always consist of the same components: neurons, synapses, weights, biases, and functions [10].

2.2 Feedforward neural networks

A feedforward neural network is an artificial neural network wherein connections between the nodes do not form a cycle [11]. It was the first and simplest type of artificial neural network devised [12]. In this network, the information moves in only in the forward direction (and hence the name): from the input nodes, through the hidden nodes (if any) and to the output nodes. There are no cycles or loops in the network. [11]

All neural networks used in this work are classes of the feedforward neural network.

2.3 Multilayer perceptron

A multilayer perceptron (MLP) is a class of feedforward artificial neural network. An MLP consists of at least three layers of nodes: an input layer, a hidden layer and an output layer. Except for the input nodes, each node is a neuron that uses a nonlinear activation function. MLP utilizes a supervised learning technique called backpropagation for training [13][14]. Its multiple layers and non-linear activation distinguish MLP from a linear perceptron. It can distinguish data that is not linearly separable [15].

The architecture of the MLPs built in this work is shown in Figure 1. It is composed of the following layers:

- Input[1] layer: this layer represent the entry point of a neural network. Its shape should be equal to the shape of the inputs. In our case the shape is determined by the number of features in the epigenomic data.
- Dense[2] layer: this is a standard fully connected layer.
- Activation[3] layer: this layer applies a certain activation function. It can be instantiated as a layer or passed as an argument to other layers (e.g. to Dense layer). Activation functions used in this work are 'PReLU'[4] (for each hidden layer) and 'ReLU'[5] (for the output layer).
- BatchNormalization[6] layer: this layer applies a transformation that maintains the mean output close to 0 and the output standard deviation close to 1 using last batch output as reference. This operation should reduce the training time and stabilize the network [16].
- Dropout[7] layer: this layer randomly sets input units to 0 with a certain (user-defined) frequency at each step during training time, which helps prevent overfitting. The Dropout layer only applies during training such that no values are dropped during inference.

[1] `https://www.tensorflow.org/api_docs/python/tf/keras/layers/InputLayer`
[2] `https://www.tensorflow.org/api_docs/python/tf/keras/layers/Dense`
[3] `https://www.tensorflow.org/api_docs/python/tf/keras/layers/Activation`
[4] `https://www.tensorflow.org/api_docs/python/tf/keras/layers/PReLU`
[5] `https://www.tensorflow.org/api_docs/python/tf/keras/layers/ReLU`
[6] `https://www.tensorflow.org/api_docs/python/tf/keras/layers/BatchNormalization`
[7] `https://www.tensorflow.org/api_docs/python/tf/keras/layers/Dropout`

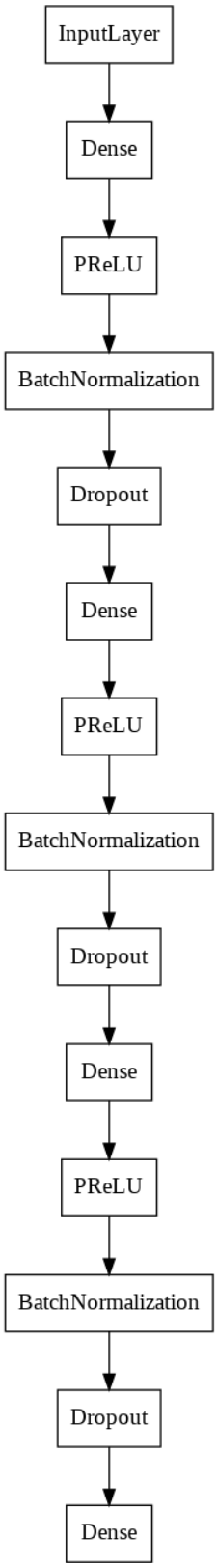

Figure 1: Multilayer perceptron architecture.

The MLPs are fed with epigenomic data, and use only this data to infer their predictions.

2.4 Convolutional neural network

Convolutional neural networks (CNNs) are a particular class of feedforward neural networks characterized by the convolution operation. This operation is performed through a kernel, called convolutional kernel, that is repetitively applied and shifted along all the input dimensions, producing the so-called feature map. Convolutional kernels are matrices that are convoluted with a portion of the input, commonly called receptive field (term borrowed from biology, more precisely from the human visual system, from which CNNs are inspired). Since these kernels are shifted along the entire input, the presence of a certain feature will be captured no matter where it occurs in the input. For this reason, CNNs are also known as shift invariant or space invariant artificial neural networks (SIANN) [17][18].

Apart from the convolution operation, CNNs are characterized by another operation called pooling. Pooling is the aggregation of features in a feature map using a certain window size juxtaposed across the entire feature map. All the data in each window are aggregated into one value, and this is tipically done to reduce the dimension of the feature maps while trying to preserve as much useful information as possible. Lowering feature maps dimension, CNNs are free to grows in depth, without much burden on the amount of computation performed in the network.

The architecture of the CNNs built in this work is shown in Figure 2. It is composed of the following layers:

- Input[8] layer: this layer represent the entry point of a neural network. Its shape should be equal to the shape of the inputs. In our case the shape is determined by the window size of nucleotides considered.
- Conv1D[9] layer: this layer creates a convolution kernel that is convolved with the layer input over a single spatial (or temporal) dimension to produce a tensor of outputs.
- Activation[10] layer: this layer applies a certain activation function. It can be instantiated as a layer or passed as an argument to other layers (e.g. to Dense layer). Activation functions used in this work are 'PReLU'[11] (for each hidden layer) and 'ReLU'[12] (for the output layer).
- MaxPooling1D[13] layer: this layer applies the pooling operation to a 1D input, using the 'max' function as aggregator.
- AveragePooling1D[14] layer: this layer applies the pooling operation to a 1D input, using the 'average' function as aggregator.
- BatchNormalization[15] layer: this layer applies a transformation that maintains the mean output close to 0 and the output standard deviation close to 1 using last batch output as reference. This operation should reduce the training time and stabilize the network [16].
- Dropout[16] layer: this layer randomly sets input units to 0 with a certain (user-defined) frequency at each step during training time, which helps prevent overfitting. The Dropout layer only applies during training such that no values are dropped during inference.
- GlobalMaxPooling1D[17] layer: same as MaxPooling1D, but it aggregates an entire feature map in one value.
- Dense[18] layer: this is a standard fully connected layer.

The CNNs are fed with sequence data, and use only this data to infer their predictions.

2.5 Multimodal neural network

A multimodal neural network (MMNN) is simply a neural network that combines different types of input data. In our case, a MMNN is built on top of a MLP and a CNN detaching the output layer of both networks, and concatenating both last hidden layers (of the MLP and the CNN) into one single layer, using the architecture shown in Figure 3.

[8] `https://www.tensorflow.org/api_docs/python/tf/keras/layers/InputLayer`
[9] `https://www.tensorflow.org/api_docs/python/tf/keras/layers/Conv1D`
[10] `https://www.tensorflow.org/api_docs/python/tf/keras/layers/Activation`
[11] `https://www.tensorflow.org/api_docs/python/tf/keras/layers/PReLU`
[12] `https://www.tensorflow.org/api_docs/python/tf/keras/layers/ReLU`
[13] `https://www.tensorflow.org/api_docs/python/tf/keras/layers/MaxPool1D`
[14] `https://www.tensorflow.org/api_docs/python/tf/keras/layers/AveragePooling1D`
[15] `https://www.tensorflow.org/api_docs/python/tf/keras/layers/BatchNormalization`
[16] `https://www.tensorflow.org/api_docs/python/tf/keras/layers/Dropout`
[17] `https://www.tensorflow.org/api_docs/python/tf/keras/layers/GlobalMaxPool1D`
[18] `https://www.tensorflow.org/api_docs/python/tf/keras/layers/Dense`

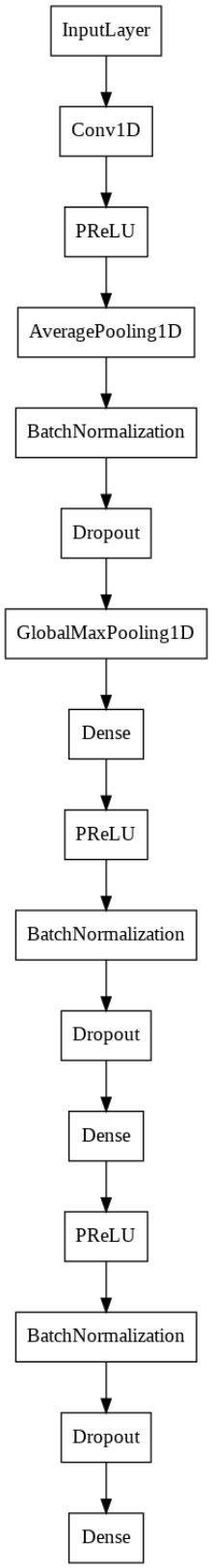

Figure 2: Convolutional neural network architecture.

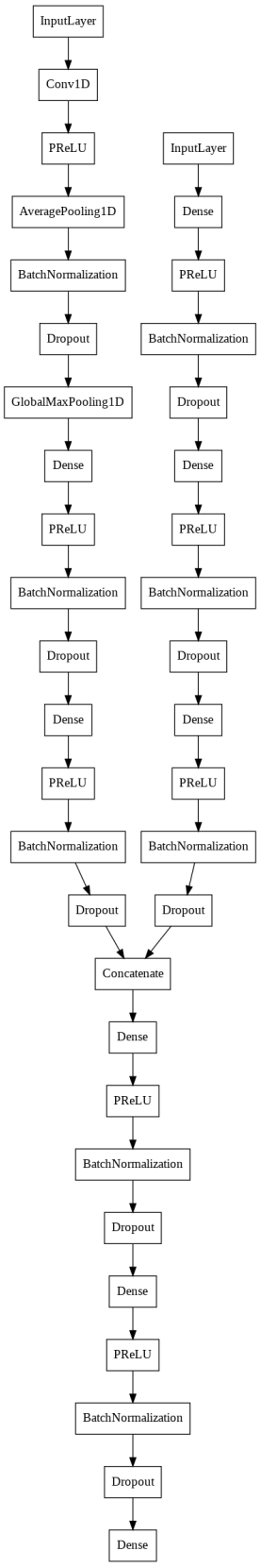

Figure 3: Multimodal neural network architecture.

The MMNN combines epigenomic and sequence data, trying to produce better results using information coming from both sources.

3 Experimental setup

The experiment is performed on the Colab[19] platform and consist in the study of two tasks:

- active enhancers vs inactive enhancers (AEvsIE),
- active promoters vs inactive promoters (APvsIP),

in the 'HEK293' immortalized cell line, on the 'HG38' dataset.

3.1 The considered tasks

The two tasks considered aim to study the activity of enhancer and promoter regions in the 'HEK293' cell line. Each task is executed using two different window sizes: 64bp and 256bp (128bp, 512bp and 1024bp are not considered due to the limited resources of the environment). Each window size determines the relative number of nucleotides associated with the enhancer or promoter region considered. For each task and for each window size, three neural network regressors are built and trained, each using one of the three architectures illustrated in the previous section. This procedure is repeated a certain number of times (in our case only two, given the limited resources of the environment used), each time randomly splitting the whole dataset in train, validation and test sets. This operation is called holdout.

3.2 Dataset source

The dataset used is composed of three parts:

- epigenomic data from ENCODE[20];
- sequence data from the UCSC Genome Browser[21];
- labels from FANTOM5[22].

ENCODE, the Encyclopedia of DNA Elements, is a public research consortium aimed at identifying all functional elements in the human and mouse genomes [19].

The UCSC Genome Browser[23] is an on-line, and downloadable, genome browser hosted by the University of California, Santa Cruz (UCSC) [21][22][23].

FANTOM (Functional ANnoTation Of the Mammalian genome) is an international research consortium led by RIKEN focused on functional annotation of mammalian genomes and characterization of transcriptional regulatory networks. In FANTOM5 the consortium, consisting of over 500 international members from 20 countries, has generated maps of human regulatory elements and transcriptional regulatory network models. [24]

Four python packages are used to select and retrieve data from these sources, in particular:

- 'epigenomic_dataset'[24] used to download epigenomic+labels data, already preprocessed for window size, from ENCODE and FANTOM5;
- 'ucsc_genomes_downloader'[25] to retrieve the human genome (assembly HG38) from the UCSC Genome Browser;
- 'keras_bed_sequence'[26] to select and one-hot encode the specific sequences related to the regions and the window sizes under study;

[19] `https://colab.research.google.com/`

[20] `https://www.encodeproject.org/`

[21] `https://genome.ucsc.edu/index.html`

[22] `https://fantom.gsc.riken.jp/5/`

[23] In bioinformatics, a genome browser is a graphical interface for display of information from a biological database for genomic data [20].

[24] `https://github.com/AnacletoLAB/epigenomic_dataset`

[25] `https://github.com/LucaCappelletti94/ucsc_genomes_downloader`

[26] `https://github.com/LucaCappelletti94/keras_bed_sequence`

- 'keras_mixed_sequence'[27] to organize and feed the data in the neural networks.

The retrieved data is composed of 99881 samples for promoter regions and 63285 samples for enhancers regions. Each sample is composed of:

- 196 features coming from the epigenomic data,
- the corresponding sequence coming from the sequence data,
- the label value.

3.3 Data preprocessing

The data preprocessing pipeline is composed of four steps:

- NaN values imputation,
- Features scaling,
- Sequence one-hot encoding,
- Labels clipping.

3.3.1 NaN values imputation

Not-a-Number (NaN) values, or missing values, are a problem from a theoretical and practical point-of-view. From a theoretical standpoint, we don't know how to interpret them, and practically speaking NaN values corrupts every subsequent calculation done on them. In order to deal with them, a KNN Imputer[28] is trained and used to impute the missing values.

3.3.2 Feature scaling

Feature scaling, or normalization, is a method used to obtain features that are comparable in their range of values, usually shifting the features to have zero mean, and rescaling them to have unit variance (or close to it). This is done because features with very different ranges of values makes learning much more difficult. In linear classifiers, for example, if a feature assumes high values, its importance is bigger in the prediction with respect to another feature with lower values. We want to avoid this and other common problems with unscaled features.

Therefore, a Robust Scaler[29] is used. As the name suggest, such scaler is robust to outliers, using quartiles to do its calculations.

3.3.3 Sequence one-hot encoding

To use sequence data, categorical values need to be converted to a better format. This is done through one-hot encoding, using the 'keras_bed_sequence' python package.

3.3.4 Labels clipping

In the field of Bioinformatics, and in particular for the tasks addressed in this work, labels can be noisy. This should not surprise, since data comes from complex biotechnological procedures. Noisy data means we may have outliers that could poison our dataset. Since we want to build regressors, ourliers may worsen their predictions. Hence, labels clipping is performed.

In order to find the optimal clipping value, we want to know how many samples are affected by each possible clipping cut. Figure 4 shows how many samples are above the clipping value for some finite values. After investigating the plots, we choose a clipping value of 50 in promoters, and a clipping value of 1 in enhancers.

Figure 5 and Figure 6 shows labels distribution before and after the clipping respectively.

[27] `https://github.com/LucaCappelletti94/keras_mixed_sequence`

[28] `https://scikit-learn.org/stable/modules/generated/sklearn.impute.KNNImputer.html`

[29] `https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.RobustScaler.html`

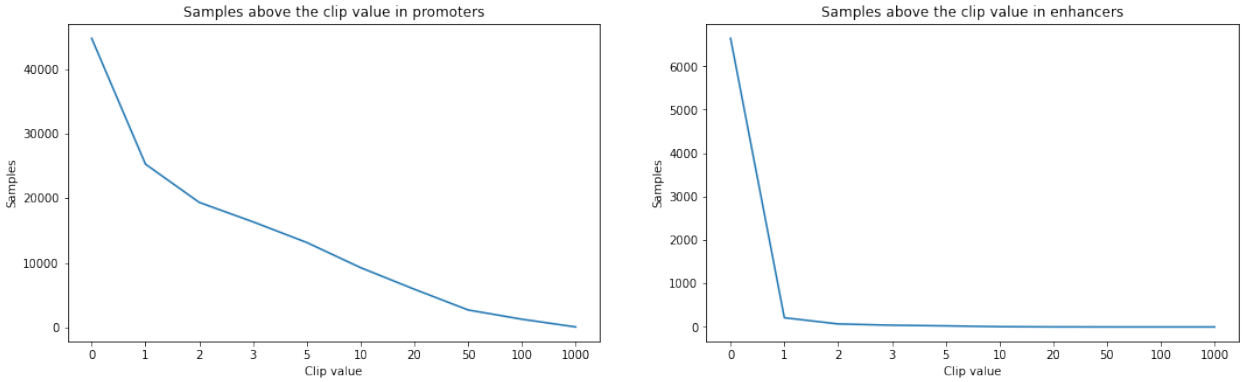

Figure 4: Number of samples above clipping values in promoters (left) and enhancers (right).

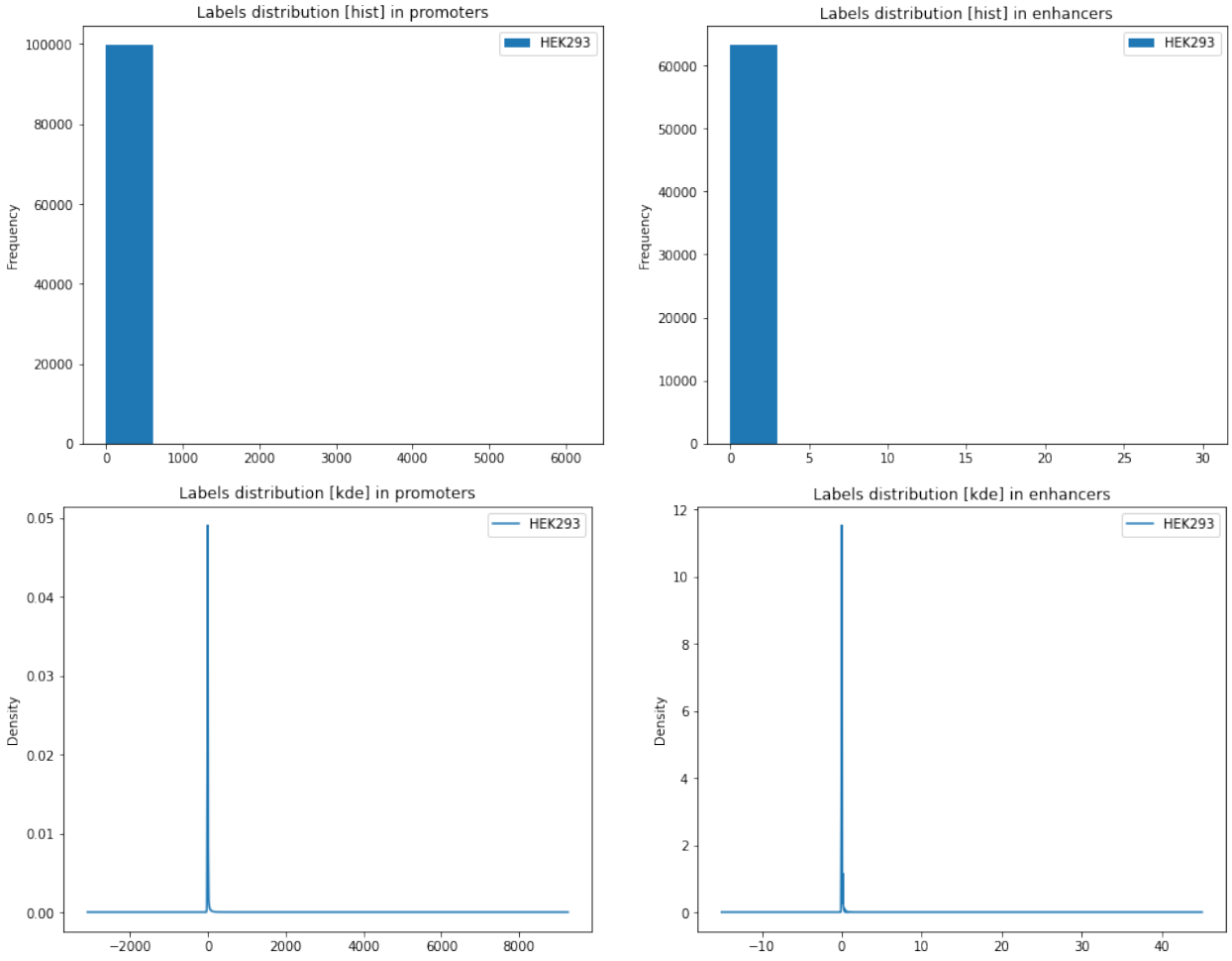

Figure 5: Distribution of labels before clipping. Histograms (top) and kernel density estimate (KDE) plots (bottom) of promoters (left) and enhancers (right).

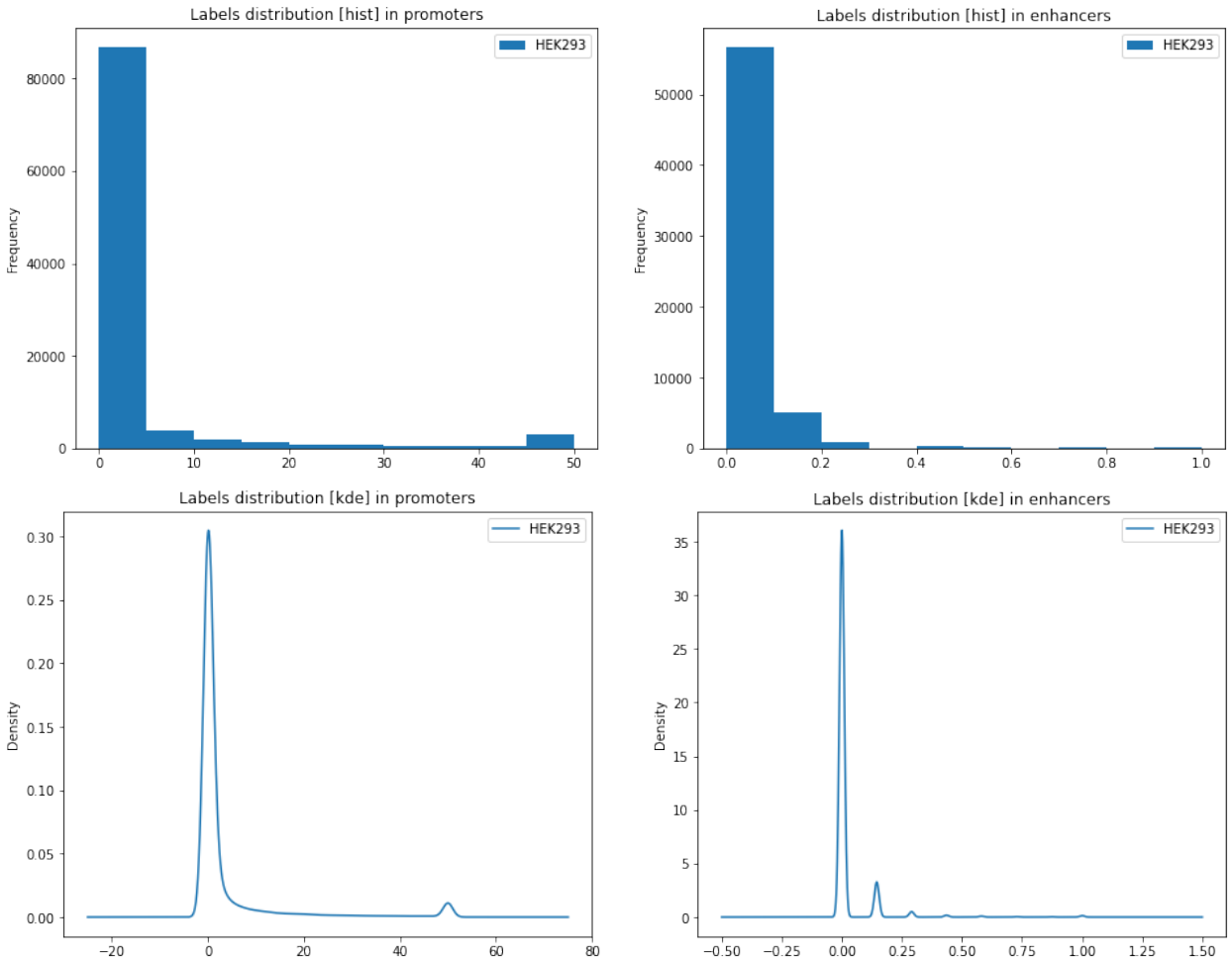

Figure 6: Distribution of labels after clipping. Histograms (top) and kernel density estimate (KDE) plots (bottom) of promoters (left) and enhancers (right).

3.4 Data correlations and distributions

Before delving in the models training, we want to explore our data. For the following exploratory analysis we will use a window size of 256bp.

3.4.1 Labels distribution

As previously stated, Figure 6 shows labels distribution.

3.4.2 Mean and max metrics comparison

Regarding the acquisition of epigenomic data, two metrics can be used to aggregate feature values coming from the window around the epigenomic regions: 'mean' and 'max' metrics. In particular, 'mean' metric aggregates values calculating the mean value in the window considered; while the 'max' metric aggregates values calculating the max value in the window considered. In general, 'mean' metric is more reliable, since values in the epigenomic data are usually noisy.

In any case, we want to understand how much these two metrics differ in term of feature values, and if it is useful to keep and merge features coming from both metrics. Using mean squared error[30] (MSE) on each feature pair, we find that all feature pairs have MSE equals to 0 on both promoter and enhancer regions except for two features: 'CTCF' and

[30] `https://scikit-learn.org/stable/modules/generated/sklearn.metrics.mean_squared_error.html`

'ZNF610'. Figure 7 and Figure 8 shows, respectively, 'CTCF' and 'ZNF610' features distributions along with their Pearson correlation[31]. We can see that only 1 feature (out of 196), namely 'CTCF', varies consistently in both regions using the two different metrics. But even this feature is extremely correlated between 'mean' and 'max' metrics. Hence, we decide to keep features coming from the 'mean' metric only.

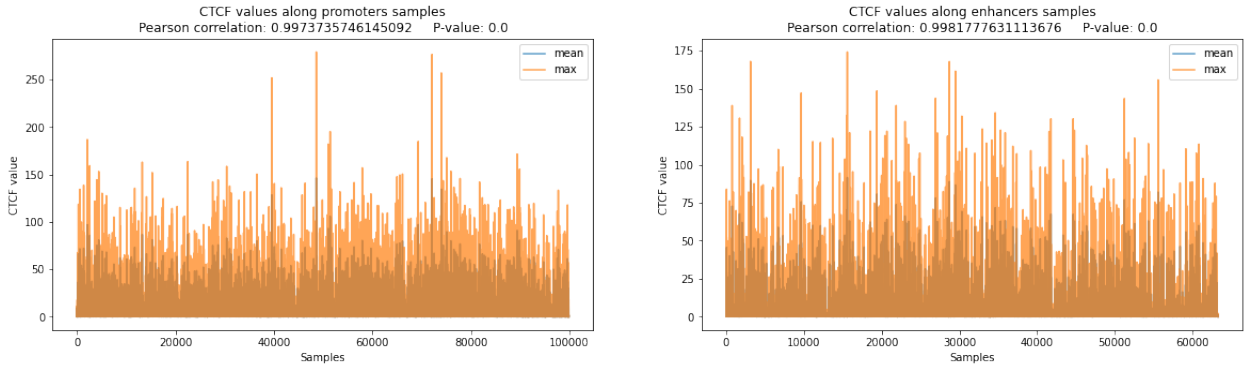

Figure 7: Distribution of 'CTCF' feature values in samples using mean and max metrics.

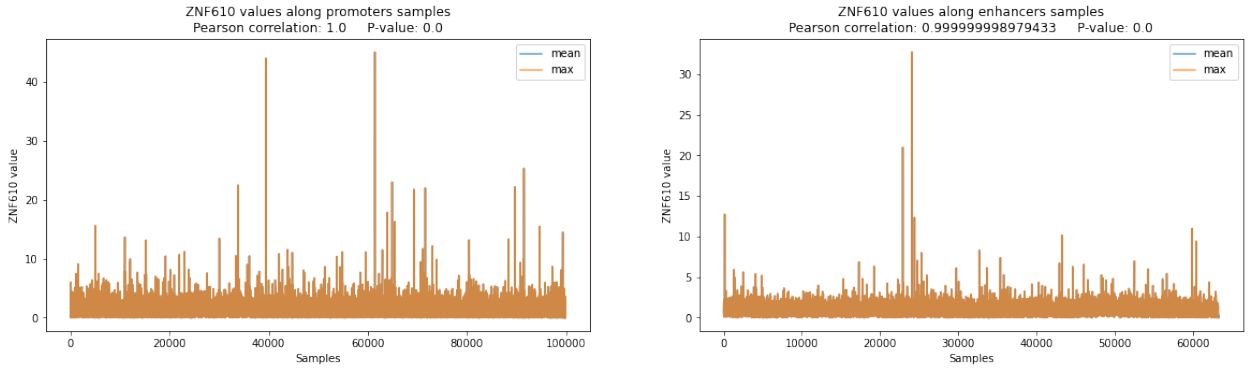

Figure 8: Distribution of 'ZNF610' feature values in samples using mean and max metrics.

3.4.3 Correlation with the output

We want to check if there are features uncorrelated with the output. If this is the case, we want to check if those features convey some information with respect to the output, using the Maximal Information-based Nonparametric Exploration (MINE). If we find out that those features don't convey any useful information for the task we are interested in, we may consider to discard them.

Table 1 shows the uncorrelated features both for promoter and enhancer regions. However, to discard features, this analysis should be repeated using the proper training set generated in each holdout split.

3.4.4 Correlation between features

We analyze if the features are pair-wise highly correlated (correlation>0.95) with statistical significance (p-value<0.05) using Pearson correlation. If it is the case, we may consider to discard one of the two correlated features since they provide almost the same information.

We show some of the most correlated features in promoters and enhancers in Figure 9 and Figure 10 respectively. Since no features highly correlated with statistical significance are found, we will keep all the features.

3.4.5 Feature distributions

Lastly, we want to show the distributions of the epigenomic features. Since there are 196 features, and we cannot show all the distributions, we select the top 10 most different features (5 for promoters and 5 for enhancers), using

[31] `https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.pearsonr.html`

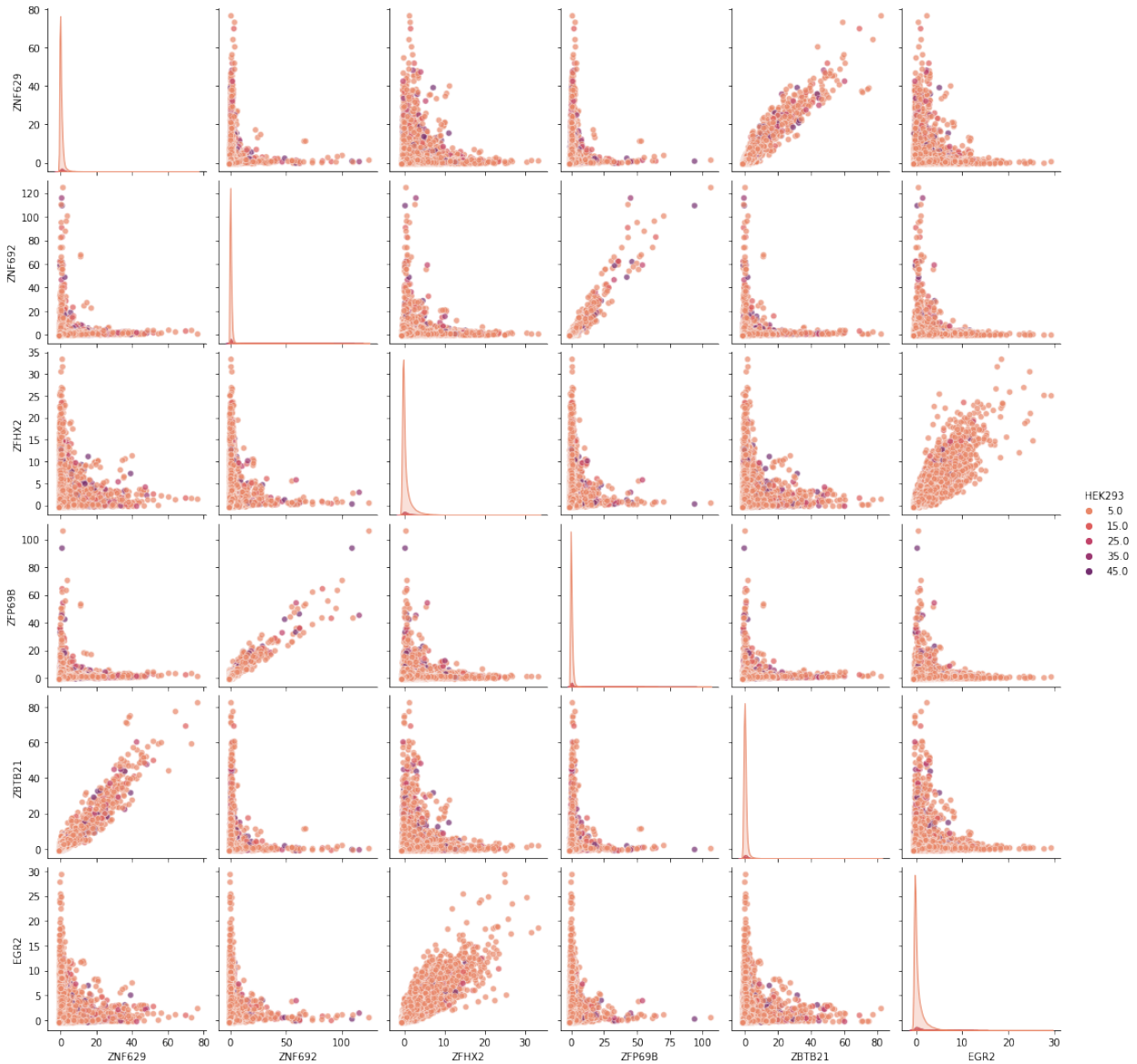

Figure 9: Plot of pairwise relationships of most correlated features in promoters. Labels are discretized in the attempt to show labels distribution.

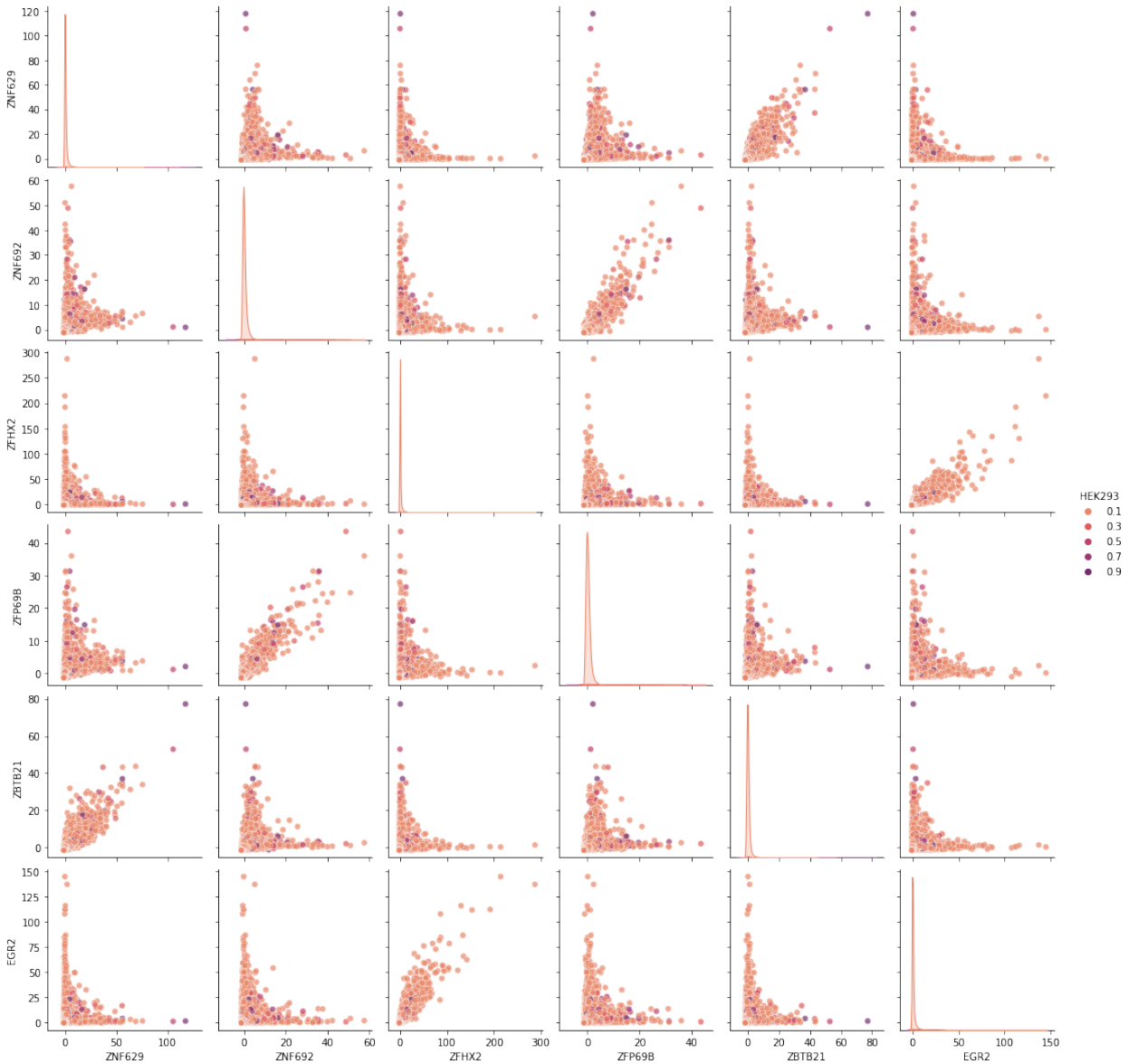

Figure 10: Plot of pairwise relationships of most correlated features in enhancers. Labels are discretized in the attempt to show labels distribution.

Region	Feature	MINE score
promoters	ZNF585B	0.006065038276248691
promoters	AEBP2	0.004943653852610387
promoters	ZNF302	0.004208120080163616
promoters	ZNF16	0.007006427852575284
enhancers	BCL6B	0.0059736457794703655
enhancers	ZNF211	0.0048443633270007945
enhancers	ZNF626	0.004733281359923767
enhancers	ZNF777	0.0051010513949622275
enhancers	ZBTB49	0.004285230044441898
enhancers	ZNF624	0.004512013351647001
enhancers	ZNF530	0.004549353125867011
enhancers	ZNF169	0.00422998087110555
enhancers	ZNF354C	0.004378372010353202
enhancers	ZNF529	0.00453861343705028
enhancers	ZNF23	0.0036573033752392795
enhancers	ZNF266	0.004519001910795977
enhancers	PRDM2	0.003744845177806166
enhancers	ZNF549	0.0044857234143696224
enhancers	ZNF768	0.003981009173558932
enhancers	ZNF426	0.0039131697960632
enhancers	ZNF404	0.004444766674471296
enhancers	ZNF548	0.004579471785392384
enhancers	ZNF680	0.004459237768615678
enhancers	ZNF701	0.004082070263454718
enhancers	ZNF677	0.005027226331470981
enhancers	RBAK	0.005185259424762877
enhancers	ZNF433	0.00452983992651429
enhancers	ZNF157	0.004877536106512652
enhancers	ZNF19	0.0044612063262768524
enhancers	ZNF791	0.004252976211764764

Table 1: The table shows the features that are considered uncorrelated with the output labels according to the Maximal Information-based Nonparametric Exploration (MINE).

the pairwise euclidean distance between the features, and represent their distributions using histograms on a log scale. Figure 11 shows the top 5 most different features for promoters and Figure 12 shows the top 5 most different features for enhancers.

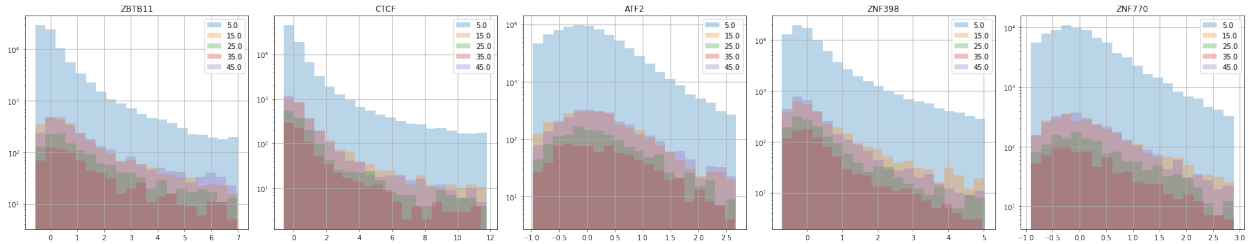

Figure 11: Distribution of the top 5 different features in promoters using histograms on a log scale. Labels are discretized to show the different labels distributions.

3.5 Feature selection

Feature selection is the process of selecting a subset of relevant features for use in model construction. The central premise when using a feature selection technique is that the data contains some features that are either redundant or irrelevant, and can thus be removed without incurring much loss of information [25]. Redundant and irrelevant are two

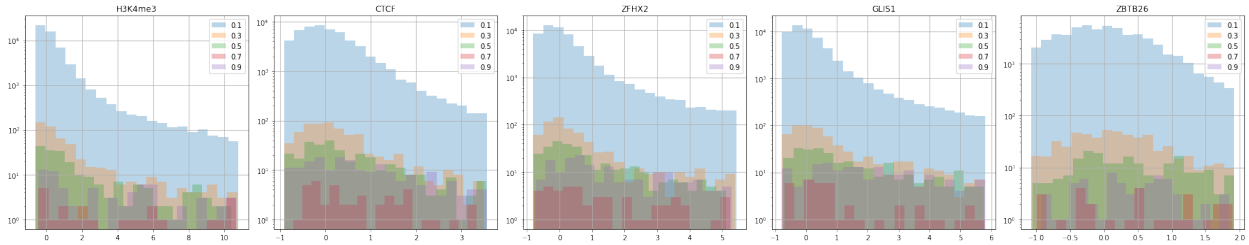

Figure 12: Distribution of the top 5 different features in enhancers using histograms on a log scale. Labels are discretized to show the different labels distributions.

distinct notions, since one relevant feature may be redundant in the presence of another relevant feature with which it is strongly correlated [26]. In the previous subsections we saw a gist of feature selection based on redundancy and irrelevance during correlation between features and correlation with the output respectively. Here we want to introduce another method to perform feature selection which is based on Random Forest: Boruta[32].

3.5.1 Boruta

Boruta is a feature selection method that trains a random forest classifier (or regressor, in our case) and applies a feature importance metric to evaluate the importance of each feature. This method should be applied during each holdout split. However, since it is computationally expensive, Boruta is skipped in this work.

3.6 Data visualization

Data visualization is a field that includes several techniques for transforming and graphing data. Here we apply a technique called 't-distributed stochastic neighbor embedding' (t-SNE) to epigenomic and sequence data.

3.6.1 t-SNE

t-SNE is a statistical method for visualizing high-dimensional data by giving each datapoint a location in a two or three-dimensional map. It is a nonlinear dimensionality reduction technique well-suited for embedding high-dimensional data for visualization in a low-dimensional space of two or three dimensions. Specifically, it models each high-dimensional object by a two- or three-dimensional point in such a way that similar objects are modeled by nearby points and dissimilar objects are modeled by distant points with high probability. [27]

Figure 13 shows the results of t-SNE applied to epigenomic and sequence data in both promoter and enhancer regions.

3.7 Holdouts

Holdout, as previously stated, is the operation of randomly splitting the whole dataset in train, validation and test sets. In this work, due to the limited resources of the environment used for the experiment, we will only perform 2 holdouts for each window size. Of course more holdouts would lead to more reliable results. In each holdout the whole dataset is subdivided in a temporary set (80% of the data) and a test set (20% of the data) using a random split. Then the temporary set is subdivided in a train set (80% of the remaining data) and in a validation set (20% of the remaining data) using again a random split.

Hence, the final proportion of data in each set is:

- 64% of the data in the train set,
- 16% of the data in the validation set,
- 20% of the data in the test set,

3.8 Results analysis

The results are built evaluating each model on the train, validation and test sets using mean squared error. Of course, since train and validation sets would lead to a biased evaluation, we will consider only the evaluation on the test set to

[32] `https://github.com/scikit-learn-contrib/boruta_py`

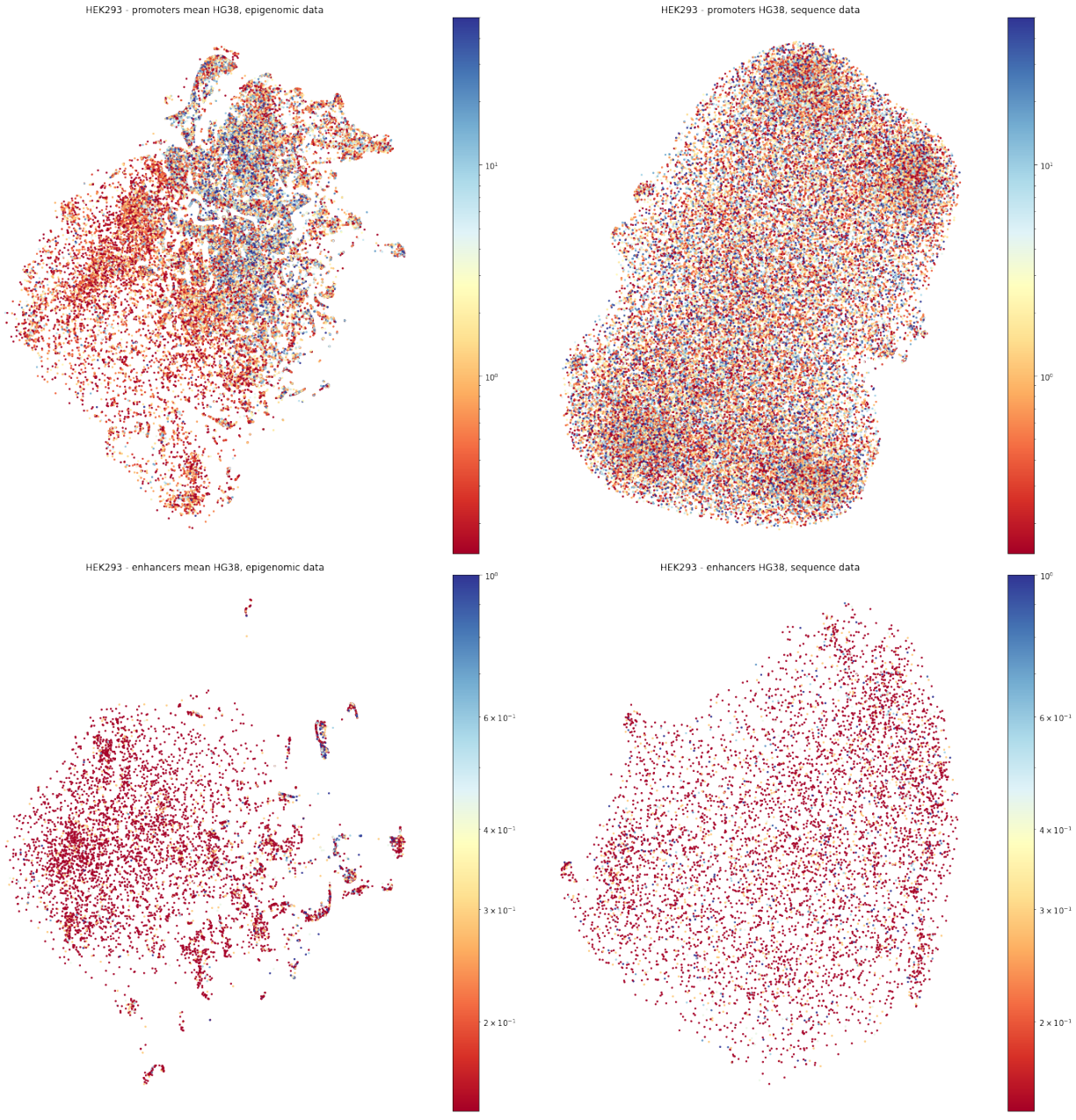

Figure 13: t-SNE of epigenomic (left) and sequence (right) data in promoter (top) and enhancer (bottom) regions.

compare performance between models and between window sizes. The Wilcoxon test is used to verify if there exists statistical differences in performance of the models and window sizes considered for the two tasks studied.

4 Results

Figure 14 shows performances on train, validation and test sets grouped by window size for the two tasks considered.

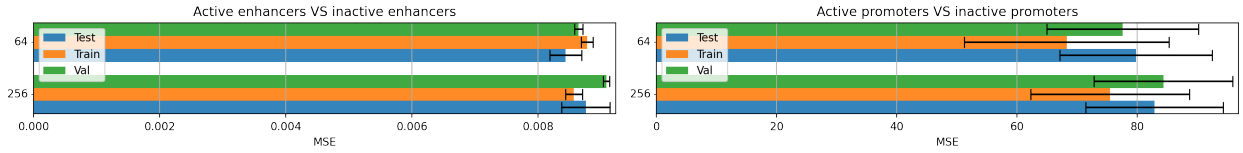

Figure 14: Barplots of train, validation and test performances grouped by window size in 'AEvsIE' (left) and 'APvsIP' (right) tasks.

Figure 15 shows test performances of models and window sizes for the two tasks considered. We can clearly see that MSE in 'AEvsIE' task is much better (lower is better) than the MSE in the 'APvsIP' task. This may be due to the different labels distributions (enhancers are clipped to 1, while promoters are clipped to 50) or to the different complexity of the two tasks (e.g. predicting the activity of promoters may be more difficult than predicting the activity of enhancers).

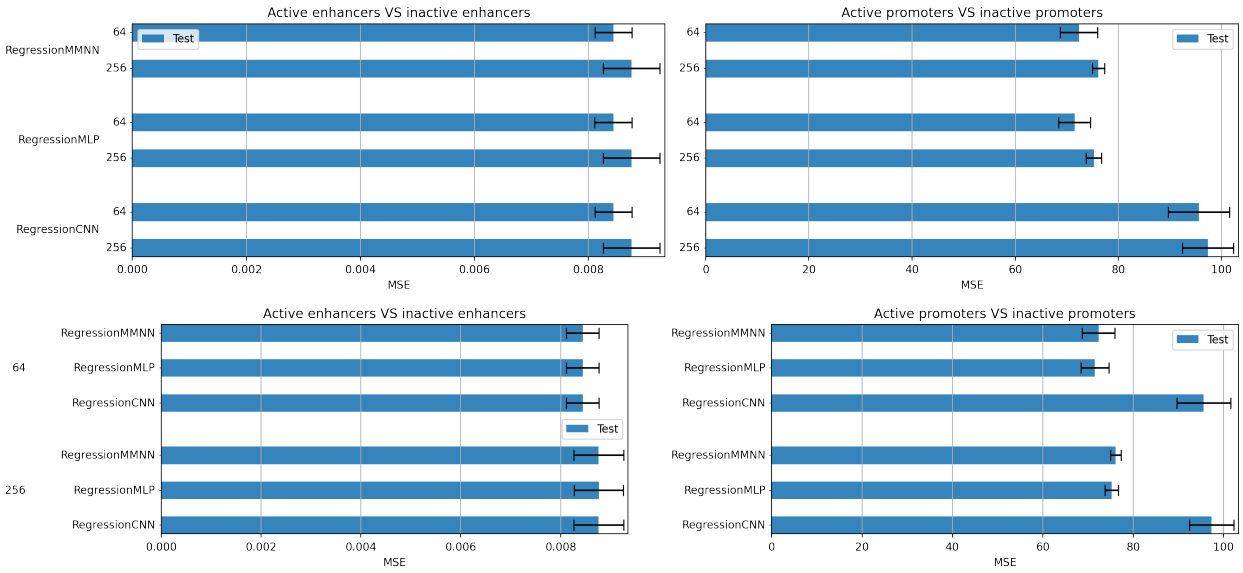

Figure 15: Barplots of test performance of models and window sizes in 'AEvsIE' (left) and 'APvsIP' (right) tasks from two different perspectives.

Lastly, Table 2 and Table 3 shows results of the Wilcoxon test for 'AEvsIE' and 'APvsIP' tasks respectively. We can see that, even if the CNN performances in 'APvsIP' seems clearly worse than MLP and MMNN performances, Wilcoxon test does not give a statistically significant difference. This is probably due to the small number of samples fed to the Wilcoxon test, that is insufficient to perform a reliable test. Nevertheless we can see that 64bp window size outperforms 256bp window size in the 'AEvsIE' task.

Task: [Active enhancers VS inactive enhancers]
- [Model] RegressionMLP is statistically indistinguishable from Regression-MMNN with p-value 0.4652088184521418 on metric mse.
- [Model] RegressionCNN is statistically indistinguishable from RegressionMLP with p-value 0.4652088184521418 on metric mse.
- [Model] RegressionCNN is statistically indistinguishable from Regression-MMNN with p-value 0.31731050786291415 on metric mse.
+ [Window size] 64bp outperforms 256bp with p-value 0.027281171477617997 on metric mse.

Table 2: Results of the Wilcoxon test for 'active enhancers vs inactive enhancers' task.

Task: [Active promoters VS inactive promoters]
- [Model] RegressionMLP is statistically indistinguishable from Regression-MMNN with p-value 0.06788915486182899 on metric mse.
- [Model] RegressionCNN is statistically indistinguishable from RegressionMLP with p-value 0.06788915486182899 on metric mse.
- [Model] RegressionCNN is statistically indistinguishable from Regression-MMNN with p-value 0.06788915486182899 on metric mse.
- [Window size] 256bp is statistically indistinguishable from 64bp with p-value 0.11585149752593009 on metric mse.

Table 3: Results of the Wilcoxon test for 'active promoters vs inactive promoters' task.

References

[1] Wikipedia. Bioinformatics. https://en.wikipedia.org/wiki/Bioinformatics. Accessed: 2021-09-03.

[2] NIH. Bioinformatics. https://www.genome.gov/genetics-glossary/Bioinformatics. Accessed: 2021-09-03.

[3] Tel Aviv University. Promoter sequence analysis. http://www.cs.tau.ac.il/~roded/courses/bnet-a06/lec11.pdf (PDF). Accessed: 2021-09-03.

[4] Elizabeth M. Blackwood and James T. Kadonaga. Going the Distance: A Current View of Enhancer Action. *Science*, July 1998.

[5] Len A. Pennacchio, Wendy Bickmore, Ann Dean, Marcelo A. Nobrega, and Gill Bejerano. Enhancers: five essential questions. *Nature Reviews Genetics*, 14(4):288–295, April 2013.

[6] Wikipedia. Immortalised cell line. https://en.wikipedia.org/wiki/Immortalised_cell_line. Accessed: 2021-09-03.

[7] Vadym M. Kavsan, Anton V. Iershov, and Olena V. Balynska. Immortalized cells and one oncogene in malignant transformation: old insights on new explanation. *BMC Cell Biology*, 12(1):23, May 2011.

[8] Yoshua Bengio. Learning Deep Architectures for AI. *Foundations and Trends® in Machine Learning*, 2(1):1–127, November 2009.

[9] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, January 2015.

[10] Yulia Gavrilova. A Guide to Deep Learning and Neural Networks. https://serokell.io/blog/deep-learning-and-neural-network-guide. Accessed: 2021-09-03.

[11] Andreas Zell, Niels Mache, Ralf Hübner, Günter Mamier, Michael Vogt, Michael Schmalzl, and Kai-Uwe Herrmann. SNNS (Stuttgart Neural Network Simulator). In Josef Skrzypek, editor, *Neural Network Simulation Environments*, The Kluwer International Series in Engineering and Computer Science, pages 165–186. Springer US, Boston, MA, 1994.

[12] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, January 2015.

[13] D. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning internal representations by error propagation. 1986.

[14] F. Rosenblatt. PRINCIPLES OF NEURODYNAMICS. PERCEPTRONS AND THE THEORY OF BRAIN MECHANISMS. 1963.

[15] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4):303–314, December 1989.

[16] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, page 448–456. JMLR.org, 2015.

[17] Wei Zhang, Kazuyoshi Itoh, Jun Tanida, and Yoshiki Ichioka. Shift-invariant pattern recognition neural network and its optical architecture. In *Proceedings of Annual Conference of the Japan Society of Applied Physics.*, 1988.

[18] Wei Zhang, Kazuyoshi Itoh, Jun Tanida, and Yoshiki Ichioka. Parallel distributed processing model with local space-invariant interconnections and its optical architecture. *Appl. Opt.*, 29(32):4790–4797, Nov 1990.

[19] NIH. The encyclopedia of dna elements. https://www.genome.gov/Funded-Programs-Projects/ENCODE-Project-ENCyclopedia-Of-DNA-Elements. Accessed: 2021-09-03.

[20] Jun Wang, Lei Kong, Ge Gao, and Jingchu Luo. A brief introduction to web-based genome browsers. *Briefings in Bioinformatics*, 14(2):131–143, March 2013.

[21] Pauline A. Fujita, Brooke Rhead, Ann S. Zweig, Angie S. Hinrichs, Donna Karolchik, Melissa S. Cline, Mary Goldman, Galt P. Barber, Hiram Clawson, Antonio Coelho, Mark Diekhans, Timothy R. Dreszer, Belinda M. Giardine, Rachel A. Harte, Jennifer Hillman-Jackson, Fan Hsu, Vanessa Kirkup, Robert M. Kuhn, Katrina Learned, Chin H. Li, Laurence R. Meyer, Andy Pohl, Brian J. Raney, Kate R. Rosenbloom, Kayla E. Smith, David Haussler, and W. James Kent. The UCSC Genome Browser database: update 2011. *Nucleic Acids Research*, 39(suppl_1):D876–D882, January 2011.

[22] W. James Kent, Charles W. Sugnet, Terrence S. Furey, Krishna M. Roskin, Tom H. Pringle, Alan M. Zahler, and and David Haussler. The Human Genome Browser at UCSC. *Genome Research*, 12(6):996–1006, June 2002.

[23] R. M. Kuhn, D. Karolchik, A. S. Zweig, T. Wang, K. E. Smith, K. R. Rosenbloom, B. Rhead, B. J. Raney, A. Pohl, M. Pheasant, L. Meyer, F. Hsu, A. S. Hinrichs, R. A. Harte, B. Giardine, P. Fujita, M. Diekhans, T. Dreszer, H. Clawson, G. P. Barber, D. Haussler, and W. J. Kent. The UCSC Genome Browser Database: update 2009. *Nucleic Acids Research*, 37(suppl_1):D755–D761, January 2009.

[24] Nature. The fantom5 project. `https://www.nature.com/collections/jcxddjndxy/`. Accessed: 2021-09-03.

[25] Anastasis Kratsios and Cody Hyndman. NEU: A Meta-Algorithm for Universal UAP-Invariant Feature Representation. *Journal of Machine Learning Research*, 22(92):1–51, 2021.

[26] Isabelle Guyon and André Elisseeff. An Introduction to Variable and Feature Selection. *Journal of Machine Learning Research*, 3(Mar):1157–1182, 2003.

[27] Wikipedia. t-distributed stochastic neighbor embedding. `https://en.wikipedia.org/wiki/T-distributed_stochastic_neighbor_embedding`. Accessed: 2021-09-03.