Prediction of transcription factor binding affinity upon cytosine methylation by using a machine learning algorithm

Treball de Fi de Grau

Author: Pau Rodríguez Riera

URV Tutor: Juan Bautista Fernández

UB Tutor: Federica Battistini

Degree: Biochemistry and Molecular Biology

Year: 2025

UNIVERSITAT ROVIRA i VIRGILI

UNIVERSITAT DE BARCELONA

I, Pau Rodríguez Riera, with DNI 49974269F, am aware of the URV plagiarism prevention guide *Prevenció, detecció I tractament del plagi en la docència: guia per estudiants*, approved in July 2017, and I affirm that this TFG does not constitute any of the behaviors considered as plagiarism by URV.

Signed,

Tarragona, 4th of June 2025

ACKNOWLEDGEMENTS

This work has been carried out in Departament de Bioquímica I Biomedicina Molecular at Universitat de Barcelona (UB), under the supervision of Professor Serra Hunter Federica Battistini, Ph.D. and Professor Juan B. Fernández Larrea, Ph.D.

I would like to thank Professor Federica Battistini, Ph.D., for helping me understand the topic and guiding me through the development of the TFG and the internship I carried at UB. Her help has not only allowed me to write this TFG but has also allowed me to hone my coding skills, my understanding of epigenetic methylation and how to critically approach and solve problems. This TFG is a representation of the future I want for myself within the bioinformatics field.

ABSTRACT

In this TFG we will assess the effect that 5-methylcytosine methylation can cause to the affinity of certain transcription factors and their corresponding transcription factor binding site. For this, we will use DNAffinity, a machine learning algorithm that predicts affinity for a specific transcription factor binding site trained on experimental data. Firstly, we will integrate and uniform different sections of the code, previously developed, in order to be able to reduce its dependencies and simplify its execution. Lastly, we will analyze several transcription factors in order to visualize the difference between both cases using different sources of experimental data like uPBM or HT-SELEX as well as adapting the features as required to allow the regressor to predict transcription factor-DNA binding affinity. The main objectives for this work are to integrate key elements for the program to work and to check if the machine learning algorithm is able to predict them correctly.

Keywords: machine learning (ML), transcription factors, methylation, data analysis.

100-WORD SUMMARY

In this TFG we will assess the effect that 5-methylcytosine methylation can cause to the affinity of certain transcription factors and their corresponding transcription factor binding site. We will use DNAffinity, a machine learning algorithm that predicts affinity for a specific transcription factor binding site trained on experimental data. This is a technical TFG, we will integrate and uniform different sections of the code, previously developed, in order allow it to work with methylation, to be able to reduce its dependencies and simplify its execution. It seems to only predict affinities correctly with HT-SELEX data.

Content Index

Acknowledgements……3

Abstract……4

Abbreviations……..7

Introduction……..8

Objectives……..14

Materials and Methods……15

Results and discussion……23

Conclusion……28

Bibliography……..29

Annex A: HT-SELEX Regressor graphs……31

Annex B: uPBM Regressor graphs……36

Annex C: uPBM PSPM, logoplot and consensus (MEME Suite vs EM algorithm)……...39

Annex D: *uPBM_Preprocess* & *selex.*py……...47

Figure Index

Figure 1……..10

Figure 2……..12

Figure 3……..16

Figure 4……..17

Figure 5……..18

Figure 6……..19

Figure 7……..23

Figure 8……..24

Figure 9……..24

Figure 10……25

Figure 11……25

Figure 12……26

ABBREVIATIONS

C-H: Carbon – hydrogen

ChIP-Seq: Chromatin Immunoprecipitation Sequencing

CpG: 5’-C-phosphate-G-3’

DNA: Deoxyribonucleic acid

dUTP: Deoxyuridine Triphosphate

EM: Expectation-Maximization

ENA: European Nucleotide Archive

GEO: Gene Expression Omnibus

GST: Glutathione S-transferase

HT-SELEX: High throughput SELEX

MEME: Multiple EM for motif elicitation

mx: slope

PBM: Protein binding microarrays

PCR: Polymerase Chain Reaction

PSPM: Position-Specific Probability Matrix

RNA: Ribonucleic acid

R0, R1,…: Round 0, 1,…

SELEX: Systematic Evolution of Ligands by Exponential Enrichment

TF: Transcription factor

TFBS: Transcription factor binding site

4mC: 4-methylcytosine

5mC: 5-methylcytosine

6mA: 6-methyladenine

INTRODUCTION

DNA is the cell's genetic information reservoir and is the core part of the dogma of molecular biology. Without DNA, proteins cannot be synthesized, which would end in complete deregulation of the organism. However, not all DNA is required on all proteins, one gene of vital importance for the correct function of a hepatocyte may not be required at all for neuronal function. This is why it is so important to regulate gene expression. There are several ways this goal can be achieved, but methylation is one of the most prevalent methods. However the means, all gene that must be expressed have to be first transcribed into RNA and, for that process to start, is where transcription factors come into play.

Transcription factors

Transcription factors (TFs) are proteins that regulate gene expression by controlling the transcription of DNA. These bind to particular DNA sequences in gene regulatory regions, called TF binding sites (TFBS), like promoters, enhancers, silencers and insulators (*1*). Although binding is required in order for these TF to work, there are also other interactions that must happen for it to exert its function, like interaction with the RNA polymerase (*2*), etc. However, we will not focus on this manner as we are only reviewing the impact on DNA binding, differentiating on whether the site is methylated or non-methylated.

According to Yumi Minyi et al. (2024) (*3*), the recognition of DNA by TF is complex and has a lot of variables that must be considered. The DNA-binding domain of a TF can selectively bind to specific sequence motifs depending on their unique *chemical signature*. This is known as a *direct readout*. Those chemical signatures can go from hydrogen bonds, methyl groups, non-polar C-H bonds and more, and must be located on the surface of the DNA's major and minor grooves to be able to be recognized by the TF. These facilitate the formation of various contact groups between the TF and the DNA. In addition to *direct readout*, the chemical signatures or the DNA nucleotides can influence the affinity between a TF and a TFBS. This is because the nucleotide sequence or modifications such as methylation influence other DNA characteristics like DNA stability, flexibility, groove with and shape (*3, 4*), which all change the number of *chemical signatures*

available for the TF to recognize and bind to, modifying its affinity. This is called *indirect readout*. However, there are also other mechanisms that can influence TF-DNA recognition, like cofactors, nucleosomes and protein-protein interactions, which further enhance the complexity of the problem. Cofactors and nucleosomes can modify chromatin environment, either allowing or disallowing TF binding and gene expression; and interactions between multiple TFs can enhance gene expression in some regions or competitively inhibit themselves, causing gene suppression.

5-methylcytosine and DNA methylation

DNA methylation is a method of epigenetic gene regulation in which methyl groups are added to DNA molecules. When added to gene regulatory regions, they can cause the region to become inaccessible to TF that usually do by altering their affinity for the region (*5*). The two only nucleobases that, as of today, have been found natural DNA methylation takes place in are cytosine and adenine, in the forms of 5-methylcytosine (5mC) (*3*, *5*, *6*), 4-methylcytosine (4mC) (*7*) and 6-methyladenine (6mA) (*8*). However, we will primarily focus on 5mC methylations since those are the ones used afterwards as affinity data for methylated TFBS. This epigenetic mark is widespread in nature, has been conserved throughout evolution and is the most common in mammals. It occurs typically at CpG sites (*3*) and is so abundant (over 80% of CpG sites are methylated (*9*)) that some papers reference it as *the fifth base* (*3*). Depending on where this methylation occurs, multiple outcomes are possible. For example, methylation of promoter regions often represses gene expression (*1*, *3*, *6*) by reducing their affinity for their usual TFs and allowing methyl-binding proteins to bind to the regions. These will then partner with histones to condense the chromatin into a transcriptionally inactive region (*1*). However, there are other specific contexts in where cytosine methylation is also associated with gene expression, like enhancer activation. Cytosine methylation in enhancers can block the affinities these regions have for their corresponding suppressors, activating enhancer function (*10*). Enhancers' main function is to boost transcription levels (*1*). They recruit TFs, co-activators and chromatin-modifying enzymes to open the chromatin and boost gene expression for a specific gene or set of genes. Some of these TFs that are

positively influenced by CpG methylation are HOXB13 (*6*), which we will use later

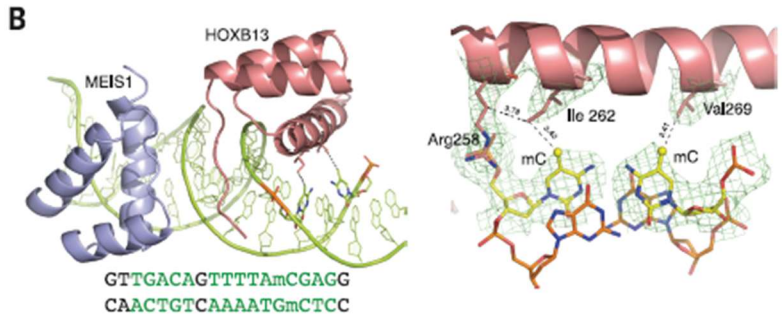


Figure 1: Overview of the HOXB13:MEIS1 heterodimer bound to a methylated DNA. HOXB13 is colored pink, MEIS1 is colored blue, the methylated base pairs are shown as ball-andstick models, the contacts are presented as dashed lines, and the residues and methylated bases are labeled. Similar to the HOXB13 monomer, the two methylated cytosines are respectively recognized by Ile262 and Val269. Image extracted from (6).

in the work to test our predictor. Therefore, even though cytosine methylation is usually in direct correlation with gene suppression, it can also be related to gene expression if some conditions are met.

Experimental data collection techniques

While this work does not use any experimental procedures itself, being a pure software work, it's important to know how the data that is used on the regressor is obtained from in order to develop the correct preprocessing code. Most data on TFBS are obtained by techniques such as PBM or SELEX experiments (*11*), which are performed *in vitro*. Specifically in this work HT-SELEX (High Throughput SELEX) and uPBM (protein binding microarrays using synthetic sequences) experimental data will be used. These techniques give a set of very short frequences, which length varies between experiments (between 10 for HT-SELEX to 40 for uPBM). There are other methods for obtaining these TFBS affinities *in vivo*, like with the use of ChIP-seq (*12*), but that's beyond the scope of this work as there is not much data available for methylated TFs and the resolution and sequence length are not optimal for algorithm training.

HT-SELEX experiments work by using a randomized library of DNA sequences that get exposed to a specific protein. Sequences that have a higher affinity with the protein will be bound more easily, forming a protein-DNA complex that can be then captured by binding them to affinity resins. For example, the protein could be marked with a hexahistidine tag, which can then bind to Ni Sepharose beads, pulling the sequences that have good affinity with them. Then the non-bound DNA is rinsed, and the bound DNA is amplified by PCR. The number of PCR cycles is

very specific and different in each experiment. A low number of PCR cycles will not suffice the amount of DNA required to run the next cycle successfully, but a high number of PCR cycles will cause the formation of bubble products due to annealing of mismatched sequences. These contain secondary structures that can be detected via gel electrophoresis, as they will appear at a higher size than their corresponding. After the PCR is done, the next cycle can be run, which will do it all over again, but with a more affinity-prone library. This means that, with each HT-SELEX cycle, the library will contain more high-affinity sequences as the ones that do not bind will be washed away. The number of HT-SELEX cycles can vary, usually from 4 to 6 cycles. In the dataset that we will use for this work (*6*), the HT-SELEX was run for 4 cycles.

To gather and analyze the results, single-end sequencing techniques like Illumina NextSeq (*13*) are used to sequence the DNA sequences that formed the protein-DNA complex to reveal their sequences. Once all DNA sequences of all cycles have been coded, they can be compared. Scripts or libraries may be used, and the idea is to count every possible k-mer in each sample and then calculate their frequency. Afterwards, they are compared between cycles to assess their affinity with the protein. For a more detailed read on the HT-SELEX protocol, all this information was gathered from recently used protocols (*6*, *14*).

uPBM experiments work by exposing a DNA microarray to a tagged protein, which will then allow us to quantify the affinity that that protein has for that specific sequence. As explained in some papers (*15*, *16*), first the microarray is designed so that it contains all possible k-mer DNA sequences synthesized as single-stranded 60-mer probe. Then those sequences are converted to double-stranded DNA and get fluorescently labeled dUTP (Cy3) to track DNA amounts. This is important because it allows us to normalize the results afterwards, as a low-affinity binding but with high amounts of DNA would appear as strong as a high-affinity with low DNA amounts. Once the microarray is ready an epitope-tagged TF is incubated in the microarray under physiological conditions, which will bind to the DNA depending on its affinity for it. Afterwards, bound TFs are labeled with a fluorophore-conjugated antibody (like Alexa 488 anti-GST), and the microarray is scanned to measure both fluorescent markers to normalize the amount of DNA

and to quantify TF binding intensity. After normalization, E-scores are assigned to each k-mer, which allows to rank them and to identify high affinity sequences.

These experiments are not only costly but have multiple points of failure (like limited number of sequences that can be tested) and take a long time to do. That's why multiple teams have changed their focus to try and find a prediction-based approach, such as DNAffinity (*11*) or others.

Random forest regressor

The software we'll use to make predictions, DNAffinity, is a random forest regressor, a form of machine learning. This technique uses an ensemble of decision trees that make predictions based on one individual feature (*17*).

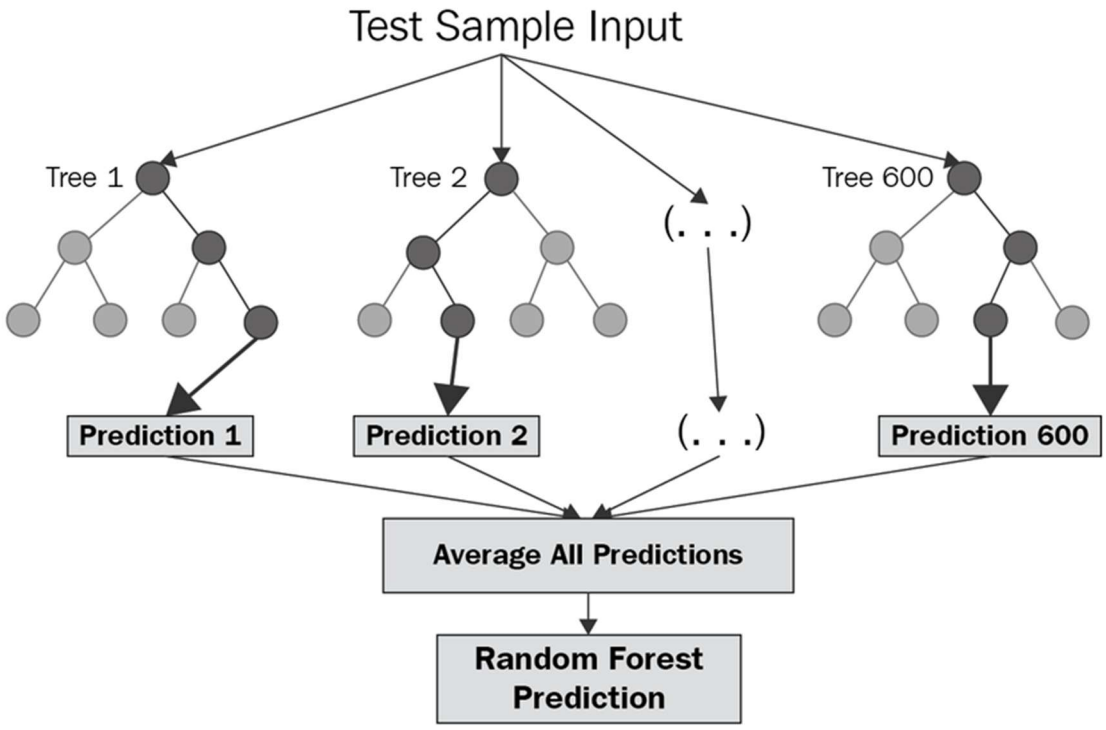


*Figure 2: Representation of a random forest regression's decision trees and how they'd work.(*18*)*

The decision tree's main job is to split the data into two groups that are as different as possible in their mean value and then uses the mean squared error

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \bar{y})^2$$

to try to minimize the variance on that value. The best split, and the one that will be kept for predictions, is the one that creates two groups that have their values

tightly clustered around their group mean. While one single tree provides a weak prediction, we can average the predictions of all the decision trees to obtain a result that will be high confidence and reliable, since that value comes from the outcome of tens or hundreds of decision trees all working on different features that affect in one way or another the prediction (*17*, *19*). This allows the random forest regressor to find correlation in non-linear relationship problems reliably. The main advantages of a random forest regressor over other machine learning models are for example its capability to handle non-linear relationships, it's robust to noise and overfitting and that random forests don't require normalization (*19*). However, it's not perfect in all situations since it's slower and requires lots of memory to run and cannot predict beyond the range of the training data (*19*).

OBJECTIVES

In this work, we will be using DNAffinity, a previously developed machine learning algorithm able to detect with high accuracy TBFS, to predict the effect of 5mC methylation on TFBS affinity. This algorithm is capable of doing so by using datasets from two different experiments: PBM (obtained from Gene Expression Omnibus GSE94634) and HT-SELEX (*6*). The Machine Learning algorithm works by using a random forest regression to predict the affinity of certain TF for DNA sequences.

The main hypothesis of the work is that DNAffinity's random forest regressor should be able to predict a TF affinity for methylated TFBS if we give it a set of features that has CpG methylation into account.

Before using this algorithm though, first we will integrate and optimize several sections of the code that depend on external sources to make sure we can obtain and manipulate the data however we see fit to ensure the best possible regression we can, and that we can see and use all data in the in-between steps.

Therefore, in this work we will:

- Understand and integrate several sections of code that would otherwise make it dependent on external pages or other programming languages.
- Verify that the new version of the code is able to predict the affinity TFs have for methylated TFBS using a new set of features.

MATERIALS AND METHODS

As stated previously, our main tool during this work will be DNAffinity. However, this work is not only a biochemistry work but a bioinformatics and therefore has a technical part. Those parts are the integration of MEME Suite (*20*) (https://meme-suite.org/meme/tools/meme) within the data preprocessing code for the uPBM random forest regressor, the translation of the R code *selex.R* into Python code for the HT-SELEX random forest regressor and a cleanup and integration of all code so it can be run automatically or with a script however we see fit. To run all the processes, we used data from the Gene Expression Omnibus GSE94634 for the uPBM raw data and the database associated with Yin Y. et al. (*6*) for the HT-SELEX raw data, extracted from ENA PRJEB9797.

MEME Suite integration

Firstly, we will start with the MEME (Multiple EM for Motif Elicitation) Suite integration. This section is within the *uPBM_Preprocess.py* code, which uses uPBM data to generate a training set for the regressor. The code first starts by loading, cleaning the experimental data and extracting DNA sequences of 35bp. Then assigns their affinity by subtracting the mean background intensity to the mean signal intensity. Afterwards, a FASTA file is created with the top 50 high-affinity sequences for motif discovery.

In the previous version of this program, this FASTA file had to be exported to MEME Suite, which uses an Expectation Maximization (EM) algorithm specially refined to detect motifs of variable or set lengths on DNA, RNA or proteins, as determined by the user. In this case, we used 10-mers as TFBS. The MEME algorithm can then display a variety of motifs and the first one (the other motifs would usually only occur in 2 or 3 sequences out of 50) was selected to explain the high-affinity sequence preferences. A Position-Specific Probability Matrix (PSPM) was then exported from the site and imported into *uPBM_Preprocess.py*. In this work we integrated this analysis into *uPBM_Preprocess.py* itself, which erases the dependance on an external webpage. We now use a custom EM algorithm. It functions by selecting a random point in all 50 sequences and extracting a 10-mer and then creating a PSPM (a matrix with the probability that a specific nucleotide is in that position). Then that PSPM is run for all possible

positions on a specific sequence, and the one that has a higher log-probability (the one that matches the best with the current PSPM) is saved as the new motif for that sequence. This step is repeated on all sequences and, once that's done, a new PSPM is generated with all those new motifs. Then this PSPM is fed into the EM loop and, once again, all sequences are run with this new PSPM to find new motif positions. This makes the motif positions more and more precise as this cycle gets repeated, until we get a PSPM that shows what the motif looks like. The number of times the EM cycle is repeated depends on the amount of data. In our case, we cycled the EM algorithm 100 times. The resulting PSPM can be used to obtain a consensus sequence. This consensus sequence can be divided into two sections: the "fixed" section, which contains the motif, and the lateral "variable" regions, which are not part of the motif and therefore can vary widely.

While our EM algorithm is simpler than a fully-fledged MEME algorithm, it still identifies the "fixed" section of the consensus sequence in most cases, although

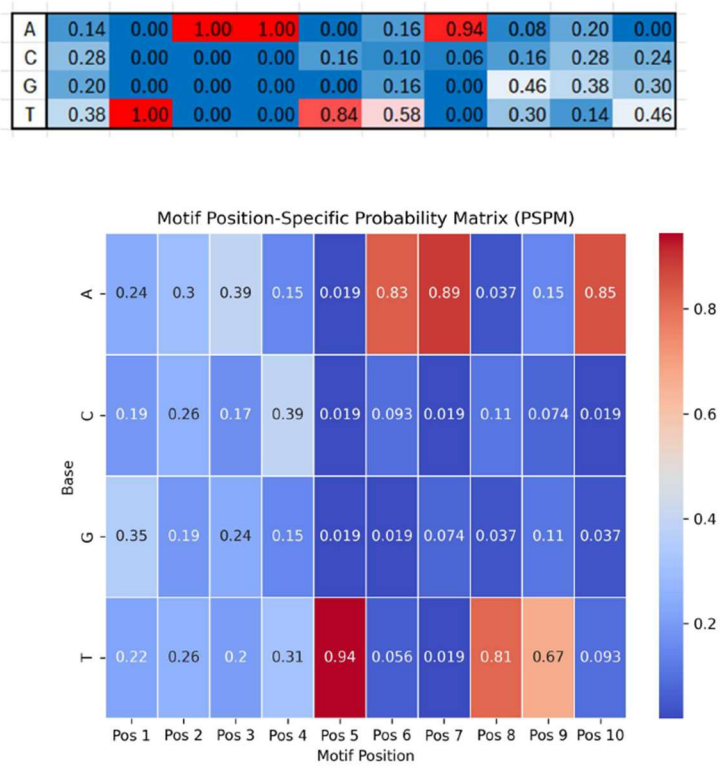


Figure 3: PSPM for the TF LHX9. Top one corresponds to the MEME Suite PSPM while the bottom one corresponds to the one from the custom EM. We can see the "fixed" motif displaced but clearly represented.

sometimes appearing displaced, as we can observe in the resultant PSPM for both MEME Suite (top) and our EM algorithm (bottom) below.

A	0.16	0.00	0.78	0.00	0.10	0.06	0.00	0.30	0.00	0.32
C	0.48	1.00	0.00	0.82	0.00	0.10	0.00	0.36	0.16	0.40
G	0.30	0.00	0.18	0.00	0.90	0.00	0.96	0.12	0.16	0.22
T	0.06	0.00	0.04	0.18	0.00	0.84	0.04	0.22	0.68	0.06

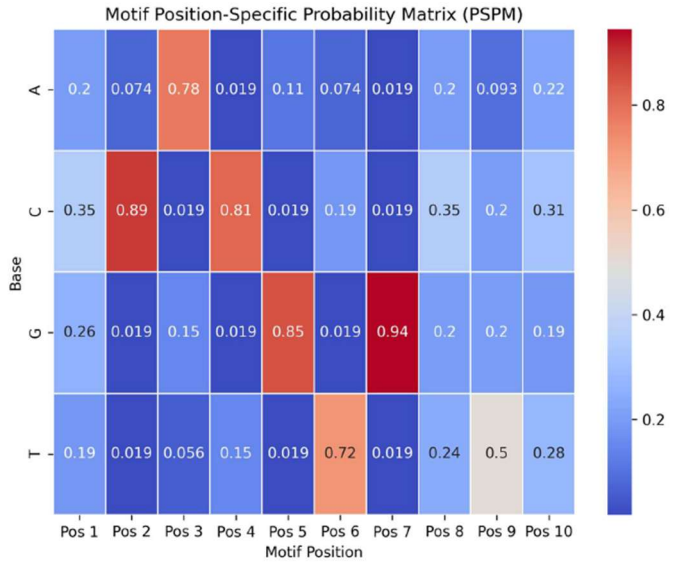


Figure 4: PSPM for the TF MAX. Top one corresponds to the MEME Suite PSPM while the bottom one corresponds to the one from the custom EM. We can see that the entire motif detected is identical position-wise, but not frequency-wise. Still, the frequencies are close enough to consider it a perfect match.

In other cases, the PSPM positions appeared to be identical regarding positions, but the frequencies appeared slightly off, although still within acceptable ranges. All PSPM comparisons are located in Annex C with their corresponding logoplots from MEME Suite.

Continuing with the execution of *uPBM_Preprocess*, the consensus sequence and PSPM are then used to align the consensus sequence to the 35-mers we extracted from the raw data and then assigns them a weight based on alignment quality. The more aligned the 35-mer is with the consensus, the more heavily weighted it will be. This is a very important step towards generating a valid training set for the regressor because a high affinity k-mer doesn't necessarily mean a strong match to the motif. This is because uPBM data is noisy due to the technique itself and we must be able to provide to the regressor a way to

understand which data is truly relevant and which one is not. High affinity by a uPBM experiment could not correlate to actual motif affinity, because of the noisy data and because affinity can vary for reasons other than the core motif, like shape. Assigning a weight to each sequence solves that issue, since it will allow the regressor to understand which data is more important to consider. The final step is to undersample. The amount of high-affinity, high-weight data we have can be extremely low compared to the large number of low-affinity sequences we have, which could bury that data causing the regressor to not be able to recognize the patterns for later prediction. Therefore, the last step is to undersample, to reduce the amount of data we will make available to the regressor.

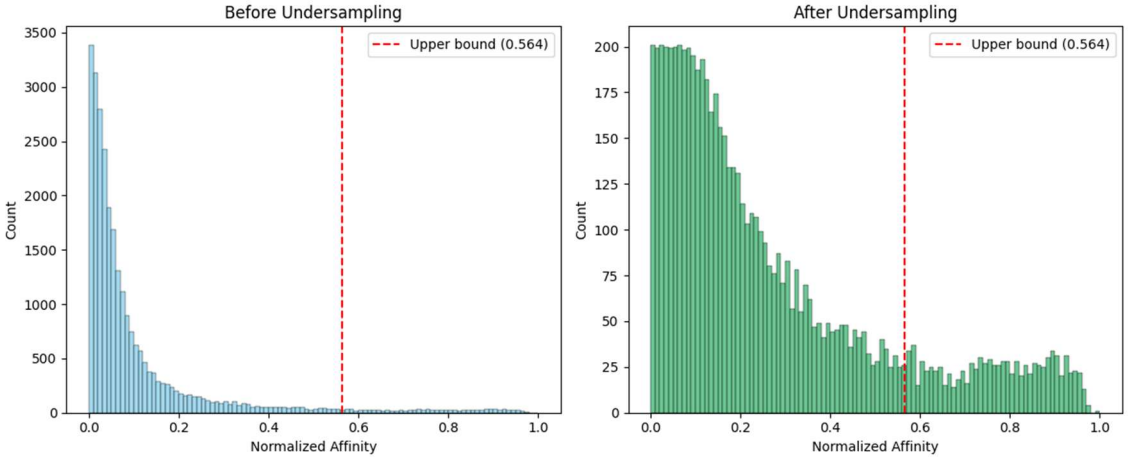


Figure 5: Undersampling represented in two graphs. As we can see on the left (not undersampled), we have an extremely high population of low-affinity sequences which won't allow the regressor to train with the high-affinity ones due to the low population of them. On the right (undersampled) we can see how the trimming of data allows the high-affinity sequences to be conserved integrally while chopping down the low-affinity irrelevant sequences.

We use two parameters for this: *L* and *upper_bound*. *upper_bound* will pick the top points as stated within the code, like in this example: *upper_bound = sort_aff[-**1000**]*. In this case, we will conserve the top 1000 points with the most affinity fully intact. The other parameter, *L*, will divide the reminder of the data into different blocks and will pick between 1 and 2 values from each, shrinking the population of low-affinity data. Once the undersampling is done the reminding, weighted data is added to a .txt file that the regressor will use to train itself.

HT-SELEX preprocessing (*selex.R*) translation

The HT-SELEX data preprocessing was mainly done by *selex.py* and allows to look for high-affinity k-mers to train the random forest regressor. In the previous version of this program, this was done by *selex*.R (which uses the R package

https://bioconductor.org/packages/release/bioc/html/SELEX.html), but it was decided to integrate it directly into python to make it easier to use, maintain and correct. The new code, *selex.py*, preprocesses the HT-SELEX data to compute k-mer affinities and their enrichment statistics. The dataset we are using (6) consists of four rounds (R0, R1, R2 and R3). R0 has the affinities for the initial random pool, while R1 to R3 contain the affinities for the subsequent enriched pool. The code works in two clear steps. The first step is to load and interpret R0 data in order to have a baseline to evaluate and quantify which k-mers are enriched in the other cycles (R1, R2 and R3). The raw data is imported in a FASTQ format and the sequences within it have their central 20-mer extracted since it contains the actual TFBS while skipping any ambiguous base (coded as N). Then these 20-mers are split between a training set (70%) and testing set (30%). The training set is then used to build a 3rd order Markov Model. A Markov Model is a complex statistical model that predicts the probability of a future event from happening based on the current state. In our context, a 3rd order Markov Model allows us to estimate the probability of a subset of 3 bases from occurring, giving us a background expectation for how likely a k-mer is to occur if there was no selection (like the R0 of a SELEX experiment). This means that the 3rd order Markov Model will be keeping track $P(base|context)$, where *P* is the probability of that event happening, *base* is the next base following the 3-mer and *context* is the 3-mer itself. To model it, the code runs a 4 base "window" through the sequence and divides it for *context* and *base* and will save those contexts and count how many occurrences are of that specific base following that specific context, and it will do that for all possible positions in all the provided sequences.

Figure 6: Visual representation of the 4-base window scanning a 6-mer, obtaining the context and base to construct the 3rd order Markov Model.

Once we have all counts for all contexts, we can convert counts to probabilities by following the formula:

$$P(base|context) = \frac{count\ of\ (base|context)}{total\ count\ of\ context}$$

and adding it to a dictionary for all contexts detected. This will return us the probabilities of a context being followed by a specific base. For example, if the context AGC was seen a total of 20 times and 15 of the bases that followed were T, 3 were A, 2 were G and 0 were C, the code will add an entry to a dictionary with the probability data which would look like this: probs = {'AGC': {'T' : 0.75, 'C': 0.15, 'A': 0.1}, ... }. With a 3rd order Markov Model ready we can now start calculating the enrichment (how often a k-mer occurs versus how often it should occur given no enrichment at all) stats for R1 to R3 in order.

First, we load the specific fastq file required (R1.fastq, R2.fastq or R3.fastq) and list all unique k-mers on all sequences, keeping track of how many times each unique k-mer appears. The code is adapted to allow the user to select the desired length of the k-mer. In this case, we went for 10-mers since it's a common number for motif finding. Then, using the 3rd order Markov Model, we can calculate the expected count for each of these k-mers by using:

$$P(kmer) = \prod_{i=0}^{k-4} P(base|context)$$

which calculates the combined probabilities of all the contexts happening in the specific k-mer from happening. With this information we can afterwards calculate the enrichment score (or affinity) for each k-mer. This is done by performing the following operations:

$$ExpectedAff = TotalKmers \cdot Prob^{Markov}$$

$$Affinity = \frac{ObservedAff}{ExpectedAff}$$

$$Standard\ Error = \frac{Affinity}{\sqrt{Count_{kmer}}}$$

This data is then put on a file and exported. The most relevant file of these will be R3, which will be saved as *3.txt* and used directly for training. Inside this file we can also see the sequences that have the most affinity with our TF, and we can check for methylation dependencies by looking at the CG within that file.

Random forest regressor

Once the training is done, we can feed the results to the regressor. The regressor should work (it did in the previous version) regardless of what experimental method was used to obtain the data.

The features are the main thing that changes in between regressors (non-methylated versus methylated). There are 4 features used by the regressor, tagged as *presence_tetramer*, *avg*, *diagonal_fce* and *electrostatic*. We can check out the original paper (*11*) to understand how it works, since the regressor has been not altered.

Firstly, the base pair parameters, like equilibrium values (*avg*) and the diagonal components of the stiffness constant matrix (*diagonal_fce*) are taken from molecular dynamics simulations covering all possible tetranucleotide contexts. The parameters include shift, slide, rise, tilt, roll and twist positions for DNA tetramers. This reflects how DNA physically behaves and not just its sequence.

Then we also consider the electrostatic pattern of each tetramer (*electrostatic*), because this models how proteins interact and recognize DNA. Each base is assigned a value based on if they are acceptors (-1), hydrophobic sites (0) or donors (+1) and then added for a tetramer perspective. For example, for 'AACT' [–1, +1, –1, 0, 0] + [–1, +1, –1, 0, 0] +[0, +1, –1, –1, +1] + [0, –1, +1, –1, 0] = [–2, 2, –2, –2, 1].

Lastly, the last feature (*presence_tetramer*) is used to generate a vector of tetramer counts for each k-mer in order to know it's composition and to be able to use the rest of the features, since they are all based on tetramers. A sliding window of 4 goes over the k-mer and keeps count of all tetramers seen. For example, the k-mer AAAAAT would yield 2 counts of AAAA, 1 count of AAAT and 0 counts for the reminder 254 possible tetramers (the total are 256 since it's the total amount of possible tetramers (4^4)).

For the methylated features, *avg* and *diagonal_fce* needed to be changed in order for them to adapt to methylated DNA. Therefore, all iterations of CG within the tetramers were changed to 5mCG data, which can be found in the paper *The*

Impact of the HydroxyMethylCytosine epigenetic signature on DNA structure and function from F. Battistini et. al. (*21*).

As for the function of the code, the random forest regressor is trained and used with the module *scikit-learn*, a popular python module used to train machine learning programs. The regressor would use 90% of the available data to train itself and the remaining 10% to test whether it could accurately predict the affinities for certain sequences regarding a specific TF. All results, be it with uPBM data or HT-SELEX data, can be found at either Annex B or Annex A respectively.

RESULTS AND DISCUSSION

Firstly, we will discuss the results obtained by the HT-SELEX regressor. After running both HT-SELEX results through the regressor (Annex A), we obtained results with an R^2 ranging from 0.45 to 0.82 for both methylated and non-methylated TF. The average was 0.57 for both too. However, we can observe some outliers above and below the mean box.

Figure 7: HT-SELEX regressor boxplot of the results of the regression for all 15 tested TF (Annex A). We can see 3 clear outliers all the way down. In general, most of the regressions have an acceptable correlation coeficient.

We can say that the top values are completely normal and not truly outliers since there are a large amount of them. What causes to be outside the box is therefore the outliers way down. Specifically, HOXB9 Cytosine (R^2 = -0.343), DMRTC2 5mC (R^2 = -0.007) and LBX2 5mC (R^2 = 0.010) all have extremely low correlation coefficients, which drag the mean down and end up excluding those top points. Therefore, the question lies on why these three TF have this low of a correlation coefficient. There are three possible answers. One is that the regressor is not working. While we can't discard this straight away because the vast amount of regressor tests (90%) have values with an acceptable range (R^2 = 0.4-0.8). The other two options are that either the data is tainted and therefore not usable or that there's no correlation between the TF and DNA and so the model cannot

predict their affinities. To see if it's the latter, we can check the logoplots located in the supplementary material S2 of (6) as well as the *3.txt* top sequences and their specific regressor affinity plot. Let's start by HOXB9 Cytosine.

If we check its *3.txt*, we can see that the top 6 sequence with the highest affinity

#	Kmer	ObservedCount	Probability	ExpectedCount	Affinity	SE
0	GCCCGACAGG	2293	4.4242639074928144e-05	124.93572666035179	18.353437093568214	0.38327931373410623
1	TACGAGGAAA	1716	3.3948123132189504e-05	95.86529015803477	17.900117938110437	0.4321129189316289
2	ATACGAGGAA	1942	4.163487046902749e-05	117.57171148059547	16.5175787231822	0.37481915095141405
3	GAGGAAACGT	2304	5.085264145384941e-05	143.60155373813046	16.044394646324914	0.33425822179843573
4	AACAACGACG	4062	9.186342902491072e-05	259.41093250114875	15.658553634712415	0.24568672130358474
5	ACGAGGAACT	1744	4.03841691211215e-05	114.0398859610761	15.292894984087052	0.36619841982213197

Figure 8: HOXB9 Cytosine 3.txt *that shows the top 6 sequences with the highest affinity. We can observe all motifs have CG somewhere within the 10-mer.*

all have the sequence CG within it. This means that HOXB9 could have a high preference for methylated sites, that wouldn't allow it to bind to regular CpG sites that are not methylated. This could mean that HOXB9, like other protein of the HOX family is sensitive to methylation and has a clear pattern in high-affinity sequences when methylated but is not present when not methylated. Therefore, our software cannot reproduce the experimental data.

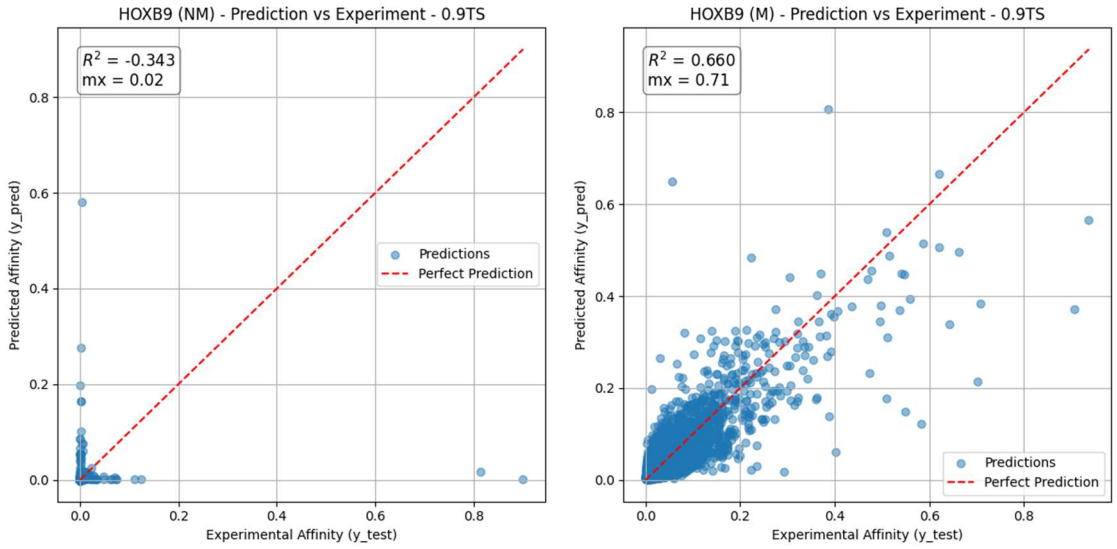


Figure 9: Regression graphs for HOXB9 in a non-methylated environment (NM) versus a methylated environment (M). We can observe how the regressor is incapable of predicting the affinities in (NM).

Further testing and study of the biological function of this TF would be required to completely justify this, but it's a plausible explanation.

LBX2 has high-affinity motifs that do not involve methylation and therefore won't influence the binding affinity significantly. If we check LBX2 5mC 3.txt, we can

observe that even the top motifs have very low affinity.

#	Kmer	ObservedCount	Probability	ExpectedCount	Affinity	SE
0	TAATTAATTA	448	8.120624404021377e-05	155.23401851976072	2.885965359087649	0.13634904700880607
1	ATTAATTATG	380	8.464301216727107e-05	161.8037513449797	2.3485240412616077	0.12047674128429754
2	TTAATTATGC	332	7.758531444700943e-05	148.31224226753213	2.2385205356218933	0.1228547750321312
3	CATTAATTAT	306	7.714863646795912e-05	147.4774877694236	2.074892952329249	0.11861373521554308
4	CATAATTAAT	383	9.774682774094798e-05	186.85303140325163	2.0497392904128984	0.10473678507546207
5	TAATTAATGA	332	8.734184236366374e-05	166.96284054606434	1.988466409137323	0.10913127195665755

Figure 10: LBX2 5mC 3.txt *that shows the top 6 sequences with the highest affinity. We can observe how they have extremely low affinity for being top sequences, especially comparing it to Figure X.*

Therefore, it makes sense that the correlation coefficient for this specific regressor run is practically null, since if it doesn't have high affinity motifs because it doesn't bind naturally to it, and therefore the regressor won't find a relation and will be unable to predict affinities reliably.

Lastly DMRTC2, although it has some high affinity sequences, they don't involve CpG sites. Therefore, methylation doesn't affect this TF.

#	Kmer	ObservedCount	Probability	ExpectedCount	Affinity	SE
0	TGTATCAATT	2012	6.649984067771463e-05	143.86814931643224	13.985027329257475	0.3117807742515275
1	AATTGATACA	2822	9.548548766900621e-05	206.5767415006841	13.660782813686964	0.25715624883645377
2	AAATTGATAC	2085	7.245273698874373e-05	156.7468594999798	13.301701907464812	0.2913091871376816
3	GTATCAATTT	1423	5.020714510570609e-05	108.61994517890837	13.10072471180289	0.34729047382215905
4	ATGTATCAAT	2185	8.133477979938806e-05	175.9625906700689	12.417412085600004	0.265647279868289
5	ATTGATACAT	2567	0.00010150042032071817	219.58966333697325	11.689985589443651	0.23072839310211207

Figure 11: DMRTC2 5mC 3.txt *that shows the top 6 sequences with the highest affinity. Although these sequences have decent affinity, the next ones fall off very quickly, especially comparing it to Figure X.*

If we check the *3.txt* further down as well, we can see that the affinity falls off pretty quickly with the rest of the motifs. Therefore, we can deduce that the regressor may have not had enough high-affinity sequences to obtain a good training to capture it, therefore making it unreliable and have a bad correlation coefficient.

However, for the rest of the points, we got a good number of points which had a good enough correlation coefficient to be deemed acceptable. This allows us to conclude that the integration of *selex.R* into the code was successful, and that it is in fact capable of predicting the affinity of TFs for 5mC methylated TFBS. A future work line could be to analyze the weight and importance of each feature used in the random forest regressor to detect and quantify the effect that

methylation causes in TF-DNA binding affinity and which TF are more dependent on 5mC methylation.

Regarding the uPBM prediction results (Annex B), the correlation coefficient is extremely low, and the results cannot be deemed valid. The average correlation

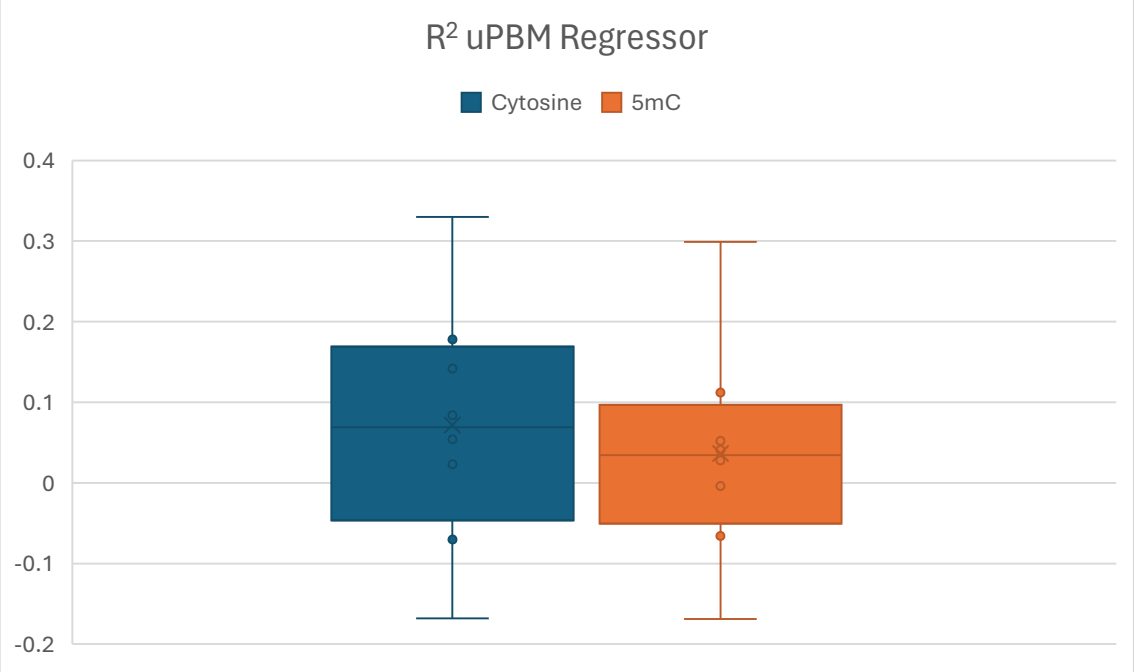


Figure 12: uPBM regressor boxplot of the results of the regression for all 8 tested TF (Annex B). The correlation coefficient is way too low to consider any correlation. The regressor was not able to predict any affinity at all.

coefficient of the data was R2 = 0.07 for the non-methylated predictions and R2 = 0.03 for the methylated one. These, of course, are too low to indicate any correlation between the regressor and the experimental data provided.

Although the integration of MEME Suite functionality with a custom EM algorithm worked in most cases, being able to detect the motif, even if displaced within the 10-mer it gave us to form the consensus, was not enough to make the regressor predict the affinity of TF-DNA binding reliably. Therefore, the integration of MEME Suite into the python code was correct, although some upgrades could be done, like actually running the EM multiple times like a MEME algorithm does in order to increase the precision of the detection of the motif. To make sure it was not the implementation of MEME Suite that was causing these errors, we also run the old *uPBM_Preprocess.py*, which required the PSPM from MEME Suite, also getting similar results. Thanks to this we can confirm that the implementation was correct. We also tried to fix it by tweaking the undersampling variables widely with

minimal improvements. This may mean that the data collected from GEO was too noisy or that there are not enough high affinity sequences available in the dataset. This would cause the regressor to not be able to obtain clear patters during its training given that it gets the training data straight from the provided dataset (*trash in, trash out*), rendering it useless and unable to predict reliably.

CONCLUSION

What we can conclude from this work is that, in most cases (90%), DNAffinity is able to predict the affinity of a TF for both methylated and non-methylated sequences with an average correlation coefficient of 0.57±0.17 if we exclude the clear outliers. This means that the translation and implementation of *selex.R* was successful, as well as the variation in the features required by the random forest regressor to be able to interpret the changes in form, electrostatic forces, etc. derived from the changes from cytosine to 5mC.

Regarding the uPBM results, as commented, the regressor seems to not be able to process the affinities correctly, given that the PSPM of both MEME Suite and *uPBM_Preprocess.py* both had the motif correctly represented within it, even if the motif was displaced. Therefore, we can state that the main objective, which was the integration of the MEME Suite algorithm, was completed successfully; but more testing and code upgrades must be done in order to make the regressor work.

Therefore, we can affirm that the code implementation was done correctly on all fronts, but that further work is required to make the regressor work again with uPBM data. Possible points of work for the future would be to analyze the importance of each feature and how 5mC methylation affect each of them, quantifying how much this epigenetic mark contributes to the decrease or increase in affinity for all studied TFs.

BIBLIOGRAPHY

1. J.-J. M. Riethoven, “Regulatory Regions in DNA: Promoters, Enhancers, Silencers, and Insulators” in *Computational Biology of Transcription Factor Binding* (2010), pp. 33–42.

2. D. S. Latchman, Transcription factors: an overview. *Int J Exp Pathol* **74**, 417–22 (1993).

3. Y. M. Yao, I. Miodownik, M. P. O’Hagan, M. Jbara, A. Afek, Deciphering the dynamic code: DNA recognition by transcription factors in the ever-changing genome. *Transcription* **15**, 114–138 (2024).

4. S. Yeou, J. Hwang, J. Yi, C. Kim, S. K. Kim, N. K. Lee, Cytosine methylation regulates DNA bendability depending on the curvature. *Chem Sci* **13**, 7516–7525 (2022).

5. R. J. Klose, A. P. Bird, Genomic DNA methylation: the mark and its mediators. *Trends Biochem Sci* **31**, 89–97 (2006).

6. Y. Yin, E. Morgunova, A. Jolma, E. Kaasinen, B. Sahu, S. Khund-Sayeed, P. K. Das, T. Kivioja, K. Dave, F. Zhong, K. R. Nitta, M. Taipale, A. Popov, P. A. Ginno, S. Domcke, J. Yan, D. Schübeler, C. Vinson, J. Taipale, Impact of cytosine methylation on DNA binding specificities of human transcription factors. *Science (1979)* **356** (2017).

7. R. A. Gaultney, A. T. Vincent, C. Lorioux, J.-Y. Coppée, O. Sismeiro, H. Varet, R. Legendre, C. A. Cockram, F. J. Veyrier, M. Picardeau, 4-Methylcytosine DNA modification is critical for global epigenetic regulation and virulence in the human pathogen *Leptospira interrogans*. *Nucleic Acids Res* **48**, 12102–12115 (2020).

8. K.-J. Wu, The epigenetic roles of DNA N6-Methyladenine (6mA) modification in eukaryotes. *Cancer Lett* **494**, 40–46 (2020).

9. M. Ehrlich, M. A. Gama-Sosa, L.-H. Huang, R. M. Midgett, K. C. Kuo, R. A. McCune, C. Gehrke, Amount and distribution of 5-methylcytosine in human DNA from different types of tissues or cells. *Nucleic Acids Res* **10**, 2709–2721 (1982).

10. J. Charlet, C. E. Duymich, F. D. Lay, K. Mundbjerg, K. Dalsgaard Sørensen, G. Liang, P. A. Jones, Bivalent Regions of Cytosine Methylation and H3K27 Acetylation Suggest an Active Role for DNA Methylation at Enhancers. *Mol Cell* **62**, 422–431 (2016).

11. S. Barissi, A. Sala, M. Wieczór, F. Battistini, M. Orozco, DNAffinity: a machine-learning approach to predict DNA binding affinities of transcription factors. *Nucleic Acids Res* **50**, 9105–9114 (2022).

12. R. Mundade, H. G. Ozer, H. Wei, L. Prabhu, T. Lu, Role of ChIP-seq in the discovery of transcription factor binding sites, differential gene regulation mechanism, epigenetic marks and beyond. *Cell Cycle* **13**, 2847–2852 (2014).

13. M. A. Quail, H. Swerdlow, D. J. Turner, Improved protocols for the illumina genome analyzer sequencing system. *Curr Protoc Hum Genet* **Chapter 18**, Unit 18.2 (2009).

14. R. Pantier, K. Chhatbar, G. Alston, H. Y. Lee, A. Bird, High-throughput sequencing SELEX for the determination of DNA-binding protein specificities in vitro. *STAR Protoc* **3**, 101490 (2022).

15. G. Badis, M. F. Berger, A. A. Philippakis, S. Talukder, A. R. Gehrke, S. A. Jaeger, E. T. Chan, G. Metzler, A. Vedenko, X. Chen, H. Kuznetsov, C.-F. Wang, D. Coburn, D. E. Newburger, Q. Morris, T. R. Hughes, M. L. Bulyk, Diversity and complexity in DNA recognition by transcription factors. *Science* **324**, 1720–3 (2009).

16. M. F. Berger, M. L. Bulyk, Universal protein-binding microarrays for the comprehensive characterization of the DNA-binding specificities of transcription factors. *Nat Protoc* **4**, 393–411 (2009).

17. L. Breiman, Random Forests. *Mach Learn* **45**, 5–32 (2001).

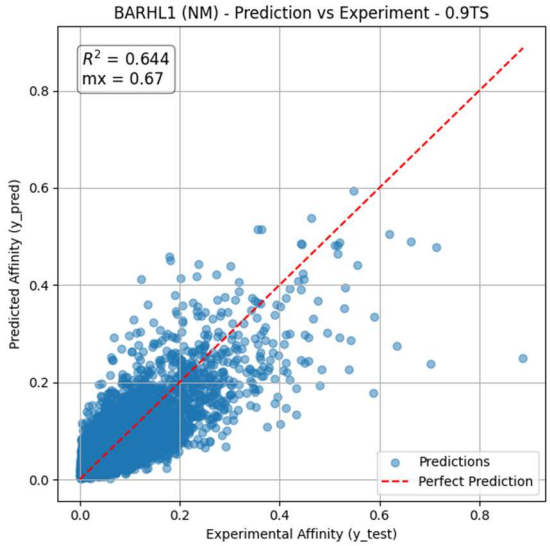
18. L. Lemeš, A. Akagic, “Prediction of Real Estate Market Prices with Regression Algorithms” (2023), pp. 401–411.

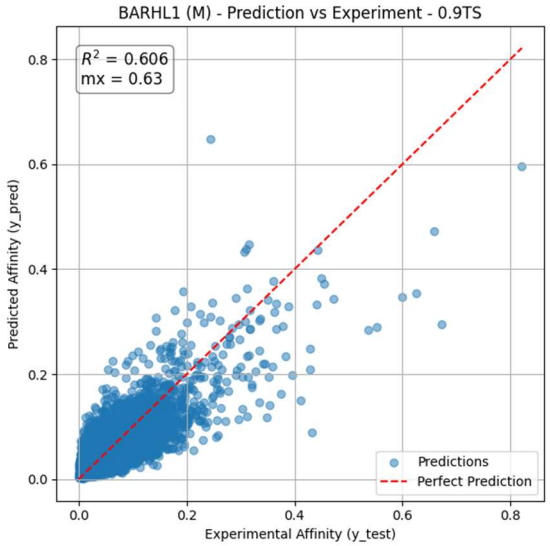
19. G. Biau, Analysis of a Random Forests Model. *Journal of Machine Learning Research* **13** (2012).

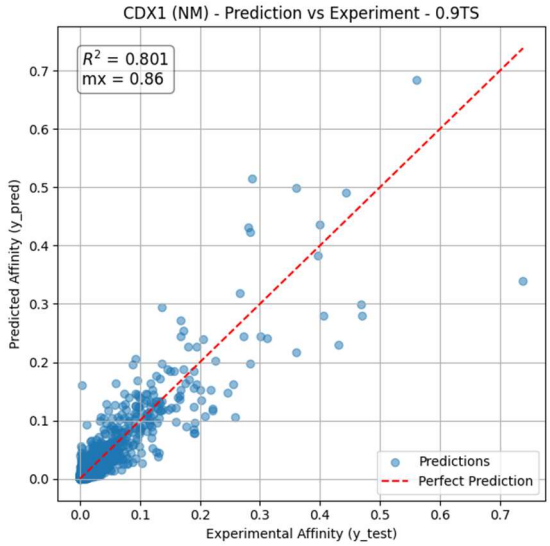
20. T. L. Bailey, C. Elkan, Fitting a mixture model by expectation maximization to discover motifs in biopolymers. *AAAI Press*, 28–36 (1994).

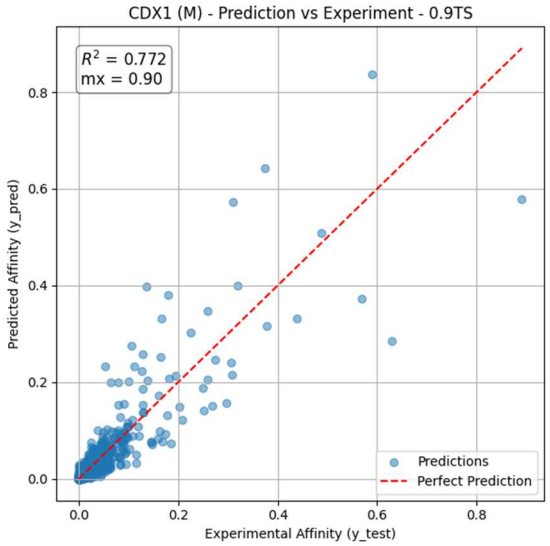
21. F. Battistini, P. D. Dans, M. Terrazas, C. L. Castellazzi, G. Portella, M. Labrador, N. Villegas, I. Brun-Heath, C. González, M. Orozco, The Impact of the HydroxyMethylCytosine epigenetic signature on DNA structure and function. *PLoS Comput Biol* **17**, e1009547 (2021).

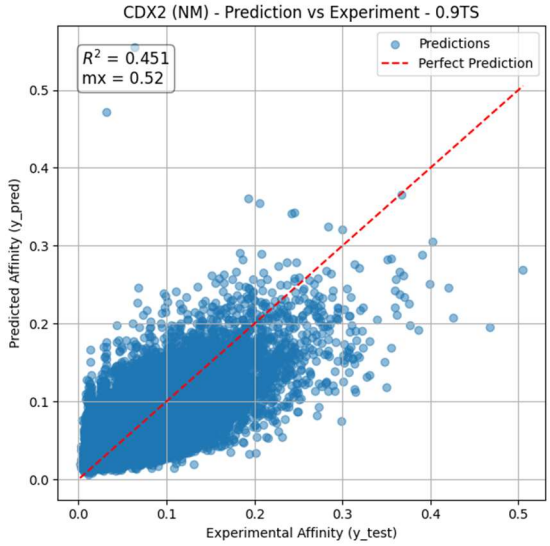
ANNEX A: HT-SELEX REGRESSOR PREDICTION RESULTS

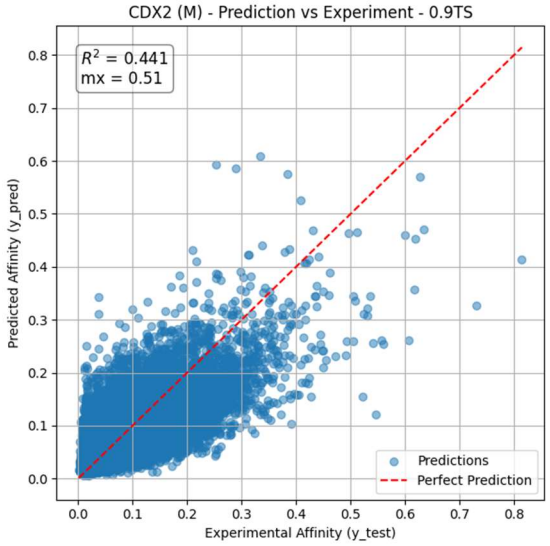


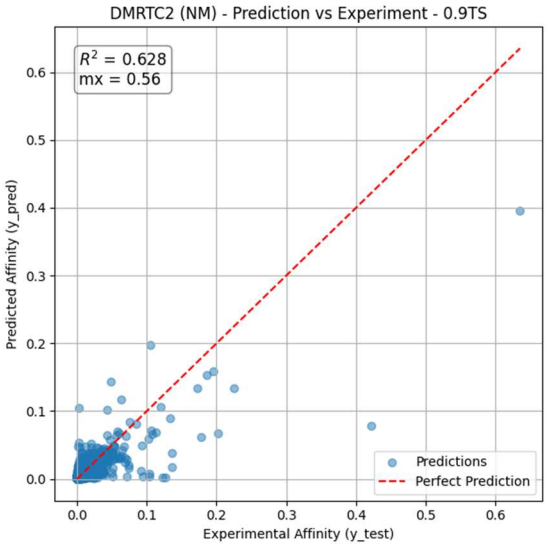


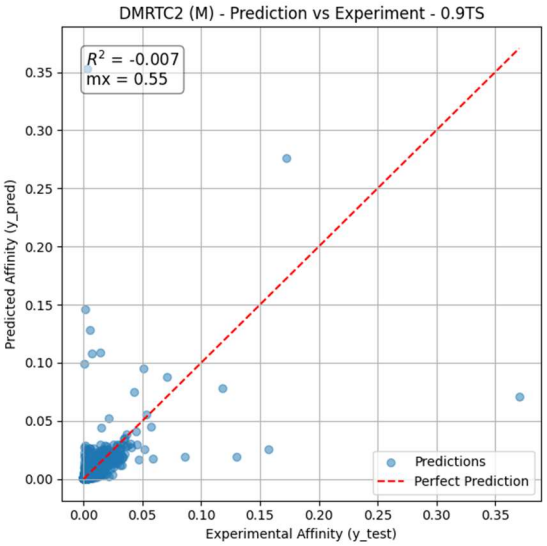


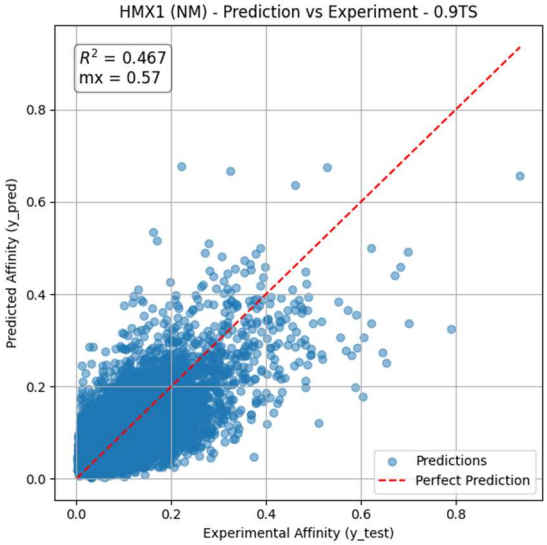


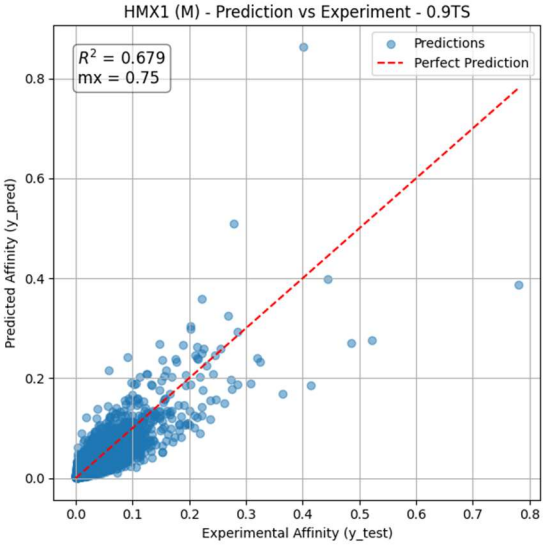


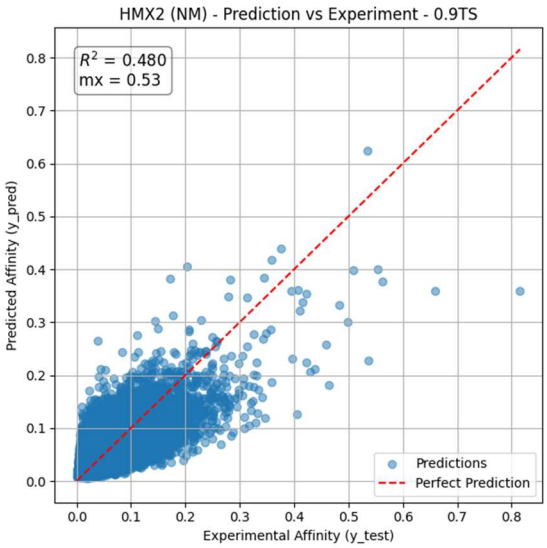


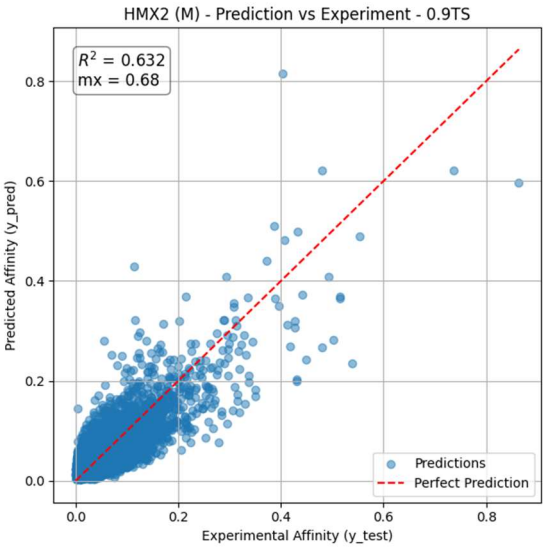

DMRTC2 (NM) - Prediction vs Experiment - 0.9TS
R^2 = 0.628
mx = 0.56
Predicted Affinity (y_pred)
Experimental Affinity (y_test)
Predictions
Perfect Prediction

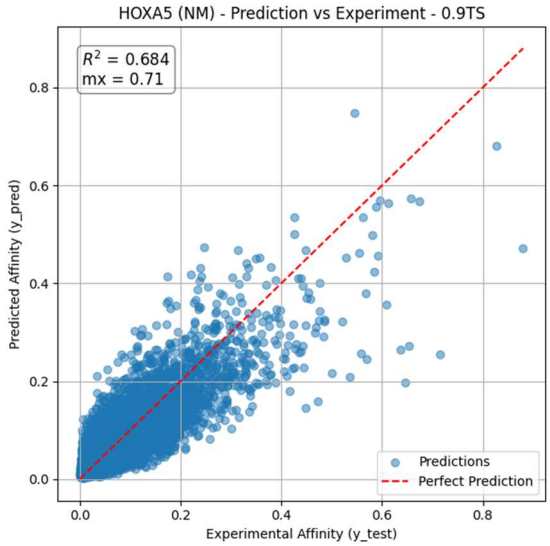

DMRTC2 (M) - Prediction vs Experiment - 0.9TS
R^2 = -0.007
mx = 0.55
Predicted Affinity (y_pred)
Experimental Affinity (y_test)
Predictions
Perfect Prediction

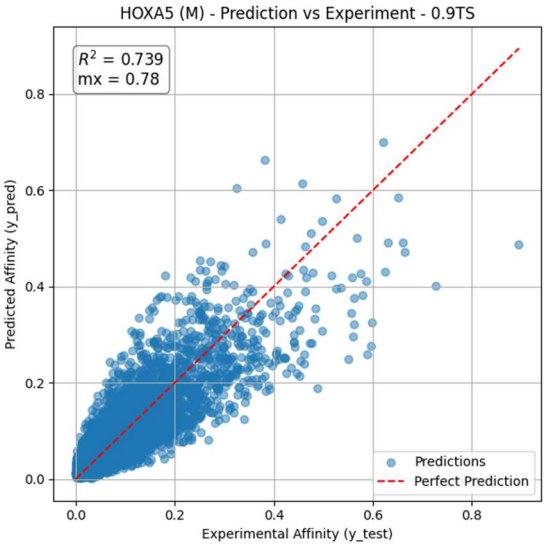

HMX1 (NM) - Prediction vs Experiment - 0.9TS
R^2 = 0.467
mx = 0.57
Predicted Affinity (y_pred)
Experimental Affinity (y_test)
Predictions
Perfect Prediction

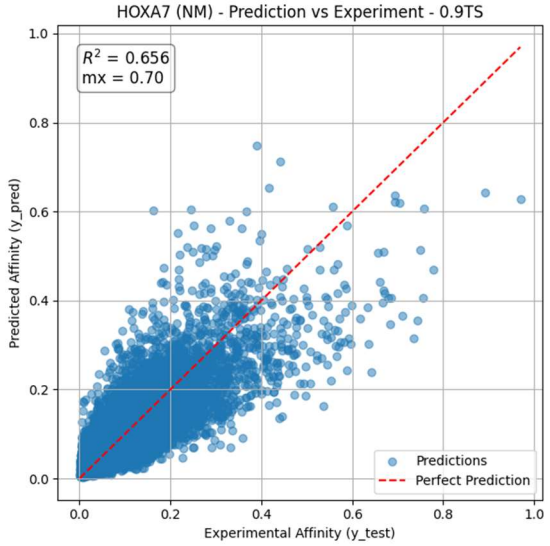

HMX1 (M) - Prediction vs Experiment - 0.9TS
R^2 = 0.679
mx = 0.75
Predicted Affinity (y_pred)
Experimental Affinity (y_test)
Predictions
Perfect Prediction

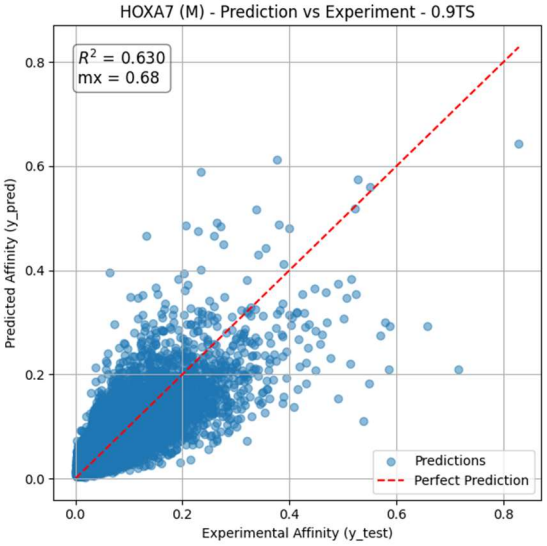

HMX2 (NM) - Prediction vs Experiment - 0.9TS
R^2 = 0.480
mx = 0.53
Predicted Affinity (y_pred)
Experimental Affinity (y_test)
Predictions
Perfect Prediction

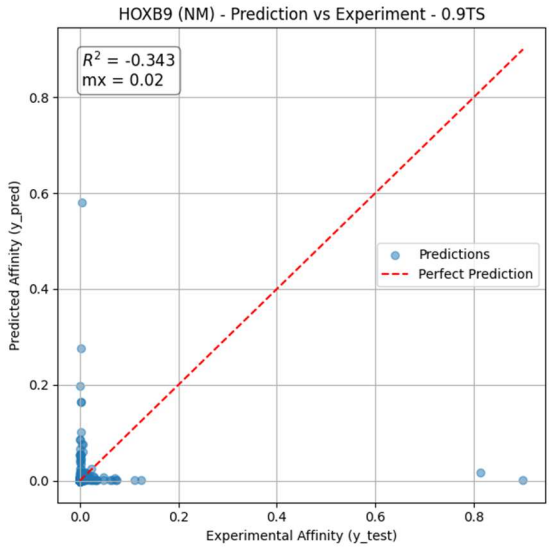

HMX2 (M) - Prediction vs Experiment - 0.9TS
R^2 = 0.632
mx = 0.68
Predicted Affinity (y_pred)
Experimental Affinity (y_test)
Predictions
Perfect Prediction

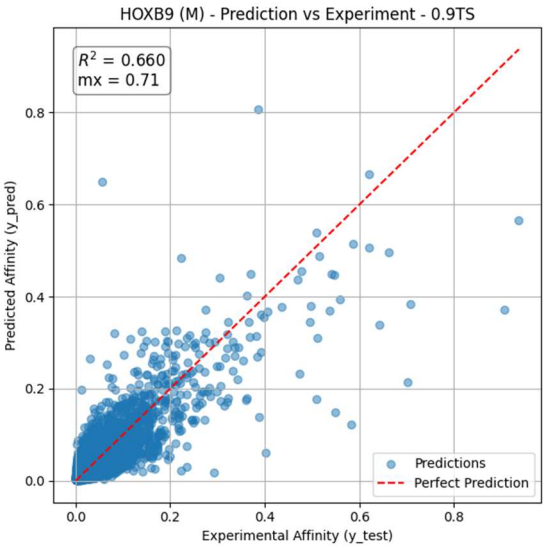

HOXA5 (NM) - Prediction vs Experiment - 0.9TS
R^2 = 0.684
mx = 0.71
Predicted Affinity (y_pred)
Experimental Affinity (y_test)
Predictions
Perfect Prediction

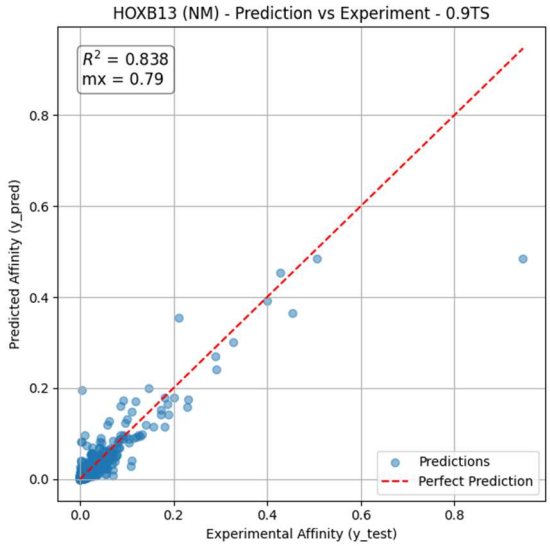

HOXA5 (M) - Prediction vs Experiment - 0.9TS
R^2 = 0.739
mx = 0.78
Predicted Affinity (y_pred)
Experimental Affinity (y_test)
Predictions
Perfect Prediction

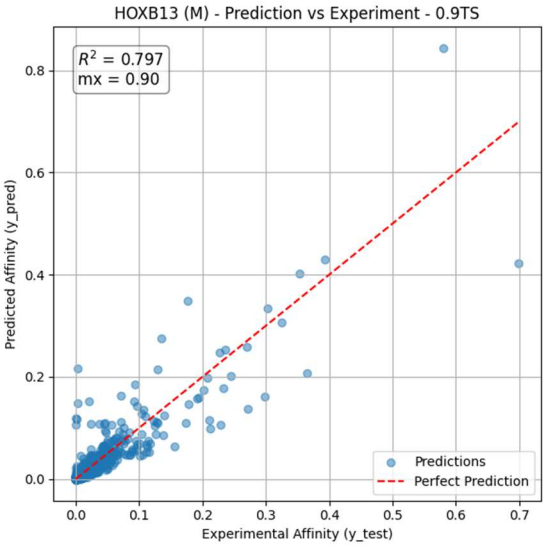

HOXA7 (NM) - Prediction vs Experiment - 0.9TS
R^2 = 0.656
mx = 0.70
Predicted Affinity (y_pred)
Experimental Affinity (y_test)
Predictions
Perfect Prediction

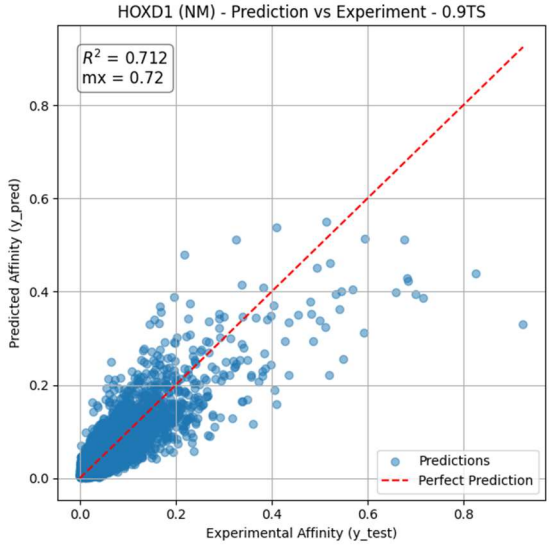

HOXA7 (M) - Prediction vs Experiment - 0.9TS
R^2 = 0.630
mx = 0.68
Predicted Affinity (y_pred)
Experimental Affinity (y_test)
Predictions
Perfect Prediction

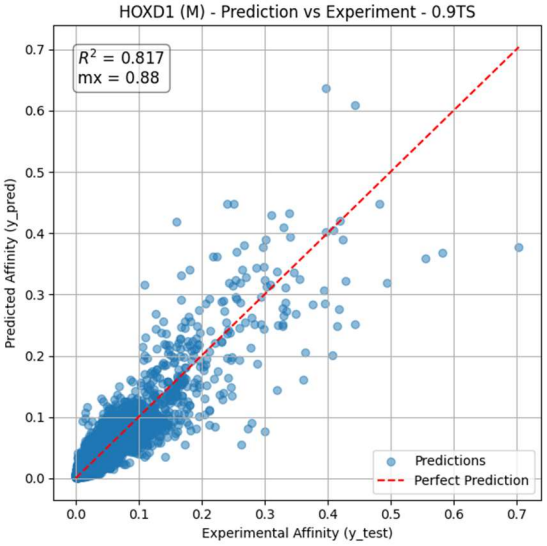

HOXB9 (NM) - Prediction vs Experiment - 0.9TS
R^2 = -0.343
mx = 0.02
Predicted Affinity (y_pred)
Experimental Affinity (y_test)
Predictions
Perfect Prediction

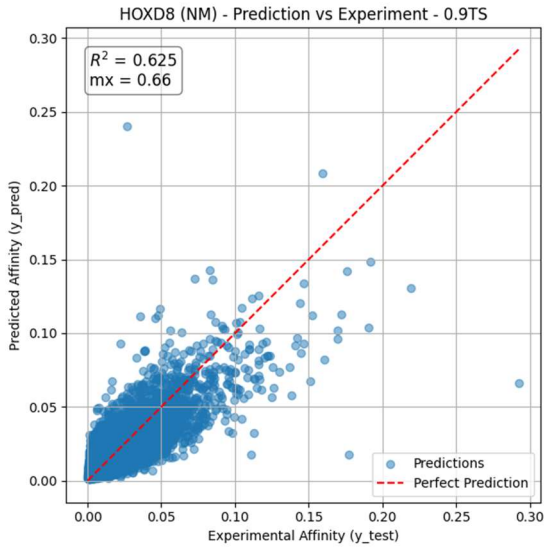

HOXB9 (M) - Prediction vs Experiment - 0.9TS
R^2 = 0.660
mx = 0.71
Predicted Affinity (y_pred)
Experimental Affinity (y_test)
Predictions
Perfect Prediction

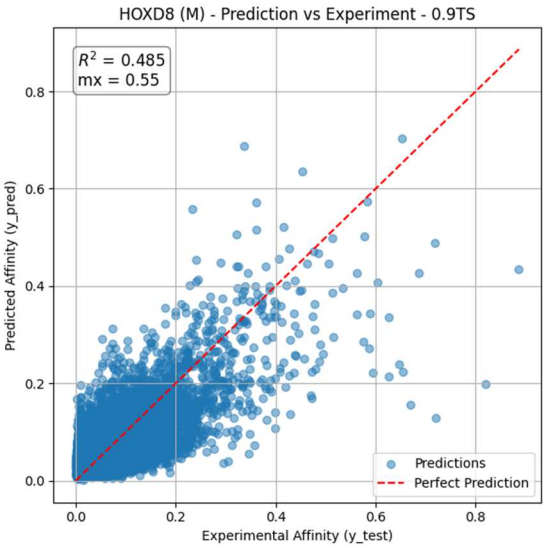

HOXB13 (NM) - Prediction vs Experiment - 0.9TS
R^2 = 0.838
mx = 0.79
Predicted Affinity (y_pred)
Experimental Affinity (y_test)
Predictions
Perfect Prediction

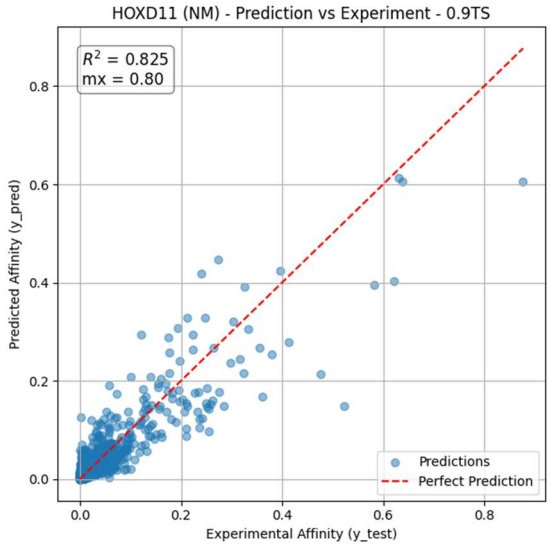

HOXB13 (M) - Prediction vs Experiment - 0.9TS
R^2 = 0.797
mx = 0.90
Predicted Affinity (y_pred)
Experimental Affinity (y_test)
Predictions
Perfect Prediction

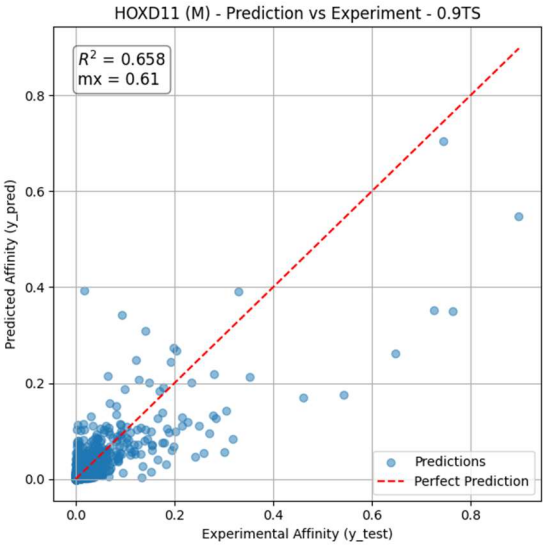

HOXD1 (NM) - Prediction vs Experiment - 0.9TS
R^2 = 0.712
mx = 0.72
Predicted Affinity (y_pred)
Experimental Affinity (y_test)
Predictions
Perfect Prediction

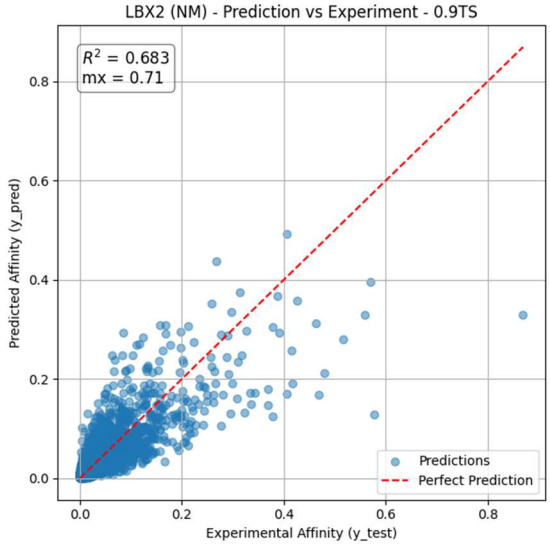

HOXD1 (M) - Prediction vs Experiment - 0.9TS
R^2 = 0.817
mx = 0.88
Predicted Affinity (y_pred)
Experimental Affinity (y_test)
Predictions
Perfect Prediction

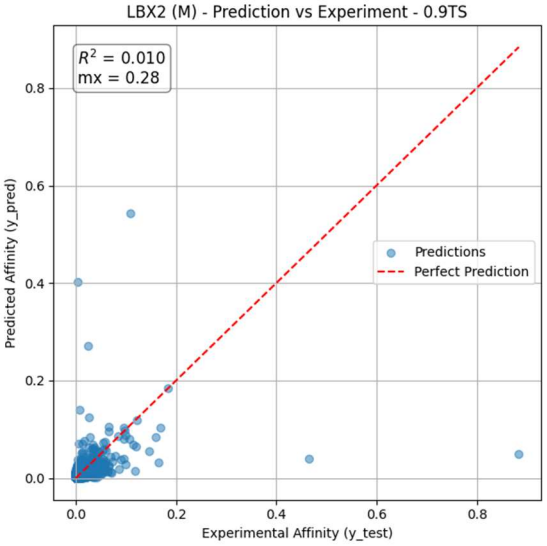

HOXD8 (NM) - Prediction vs Experiment - 0.9TS
R^2 = 0.625
mx = 0.66
Predicted Affinity (y_pred)
Experimental Affinity (y_test)
Predictions
Perfect Prediction

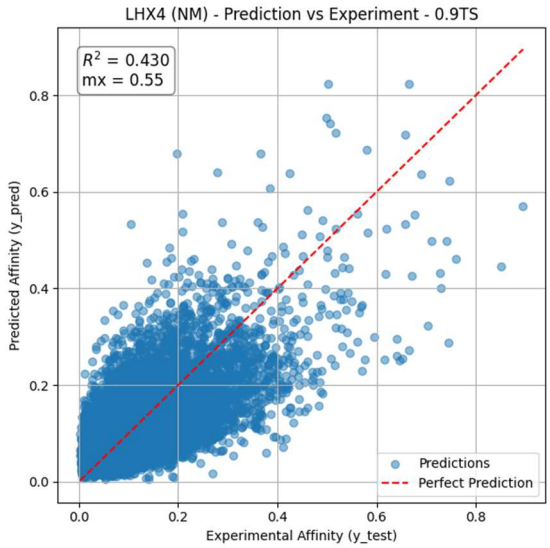

HOXD8 (M) - Prediction vs Experiment - 0.9TS
R^2 = 0.485
mx = 0.55
Predicted Affinity (y_pred)
Experimental Affinity (y_test)
Predictions
Perfect Prediction

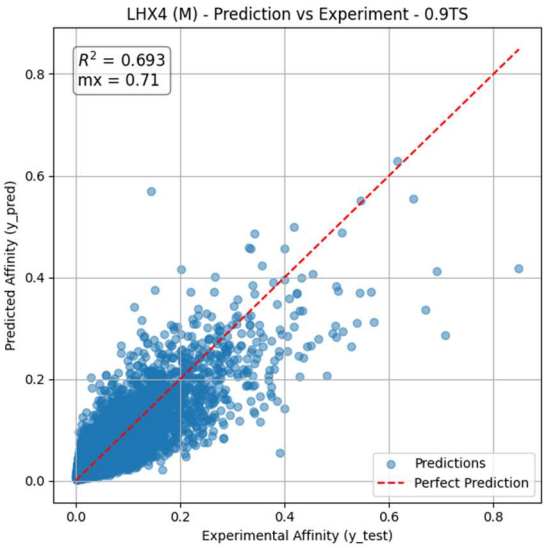

HOXD11 (NM) - Prediction vs Experiment - 0.9TS
R^2 = 0.825
mx = 0.80
Predicted Affinity (y_pred)
Experimental Affinity (y_test)
Predictions
Perfect Prediction


HOXD11 (M) - Prediction vs Experiment - 0.9TS
R^2 = 0.658
mx = 0.61
Predicted Affinity (y_pred)
Experimental Affinity (y_test)
Predictions
Perfect Prediction

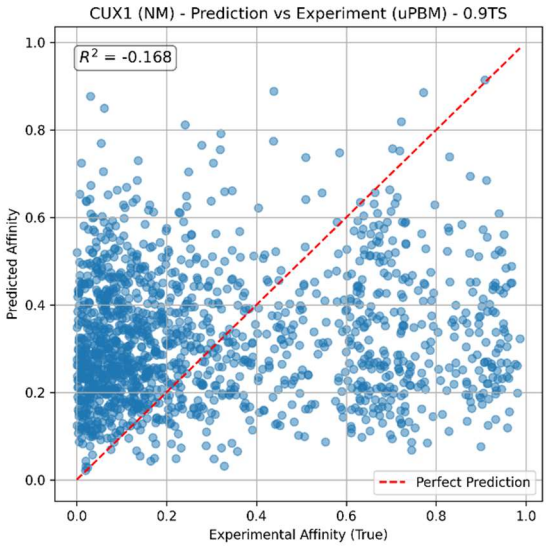

LBX2 (NM) - Prediction vs Experiment - 0.9TS
R^2 = 0.683
mx = 0.71
Predicted Affinity (y_pred)
Experimental Affinity (y_test)
Predictions
Perfect Prediction

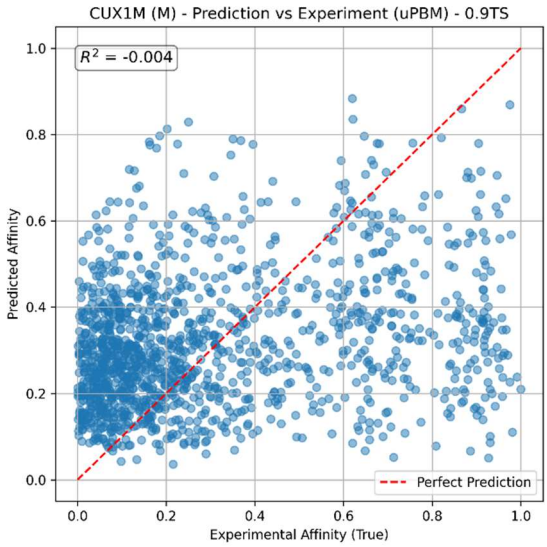

LBX2 (M) - Prediction vs Experiment - 0.9TS
R^2 = 0.010
mx = 0.28
Predicted Affinity (y_pred)
Experimental Affinity (y_test)
Predictions
Perfect Prediction

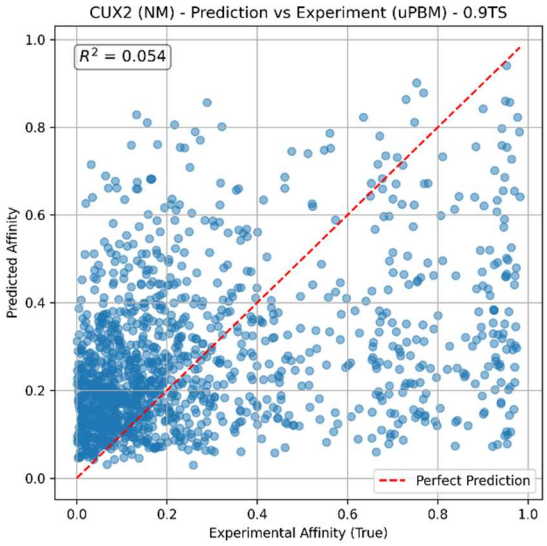

LHX4 (NM) - Prediction vs Experiment - 0.9TS
R^2 = 0.430
mx = 0.55
Predicted Affinity (y_pred)
Experimental Affinity (y_test)
Predictions
Perfect Prediction

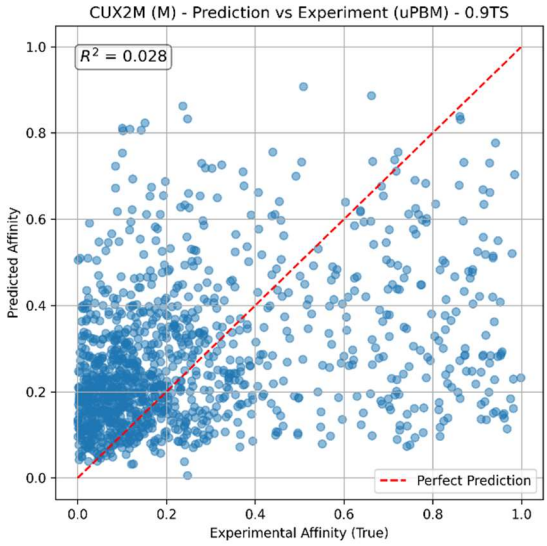

LHX4 (M) - Prediction vs Experiment - 0.9TS
R^2 = 0.693
mx = 0.71
Predicted Affinity (y_pred)
Experimental Affinity (y_test)
Predictions
Perfect Prediction

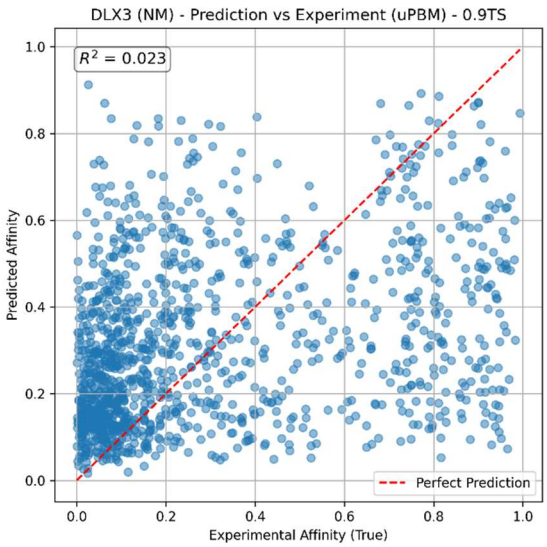
ANNEX B: uPBM REGRESSOR PREDICTION

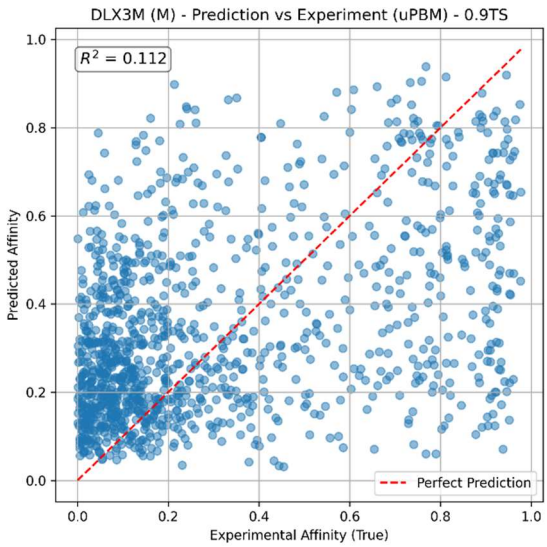


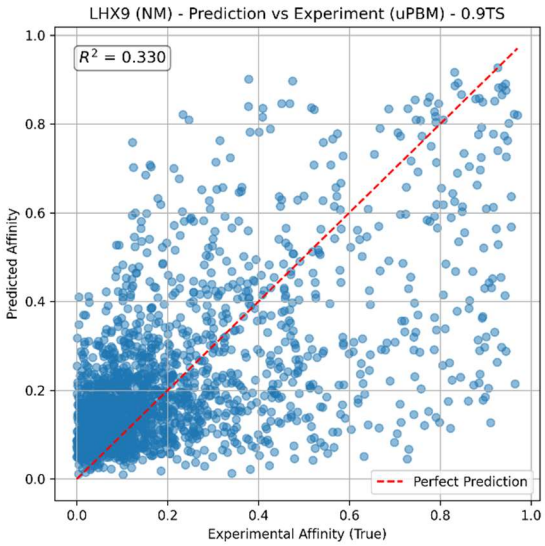


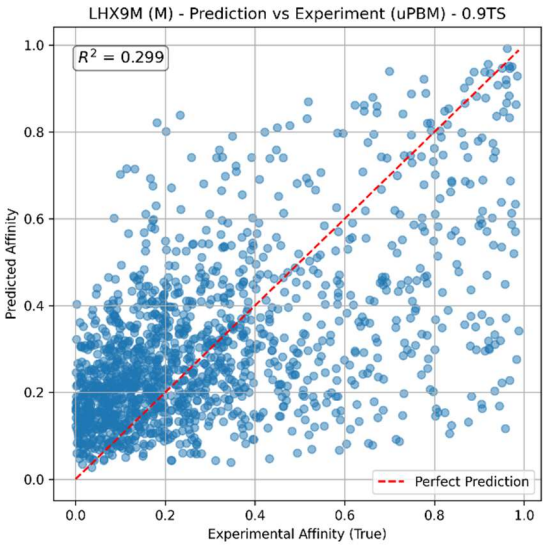


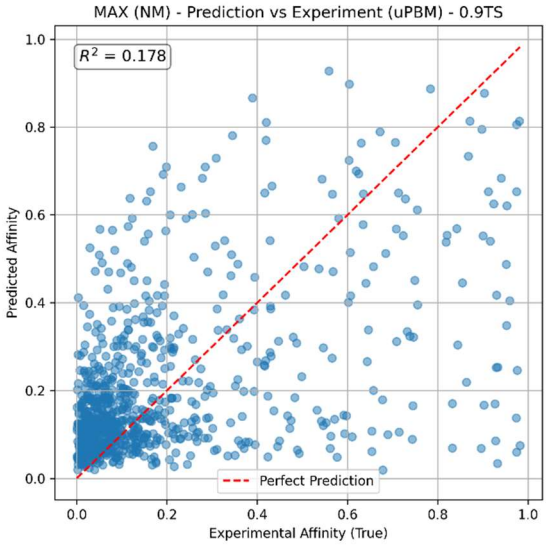


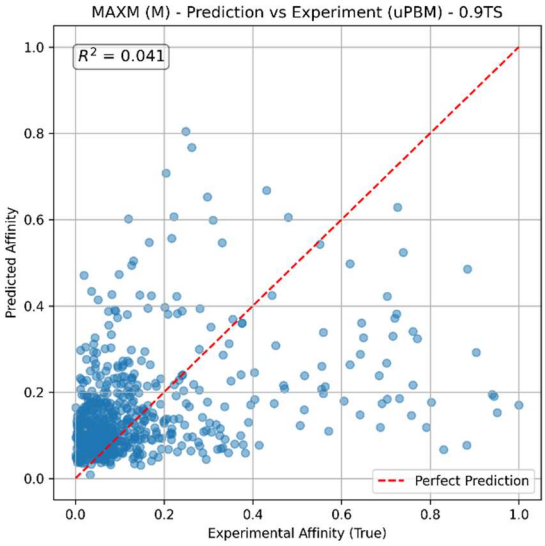


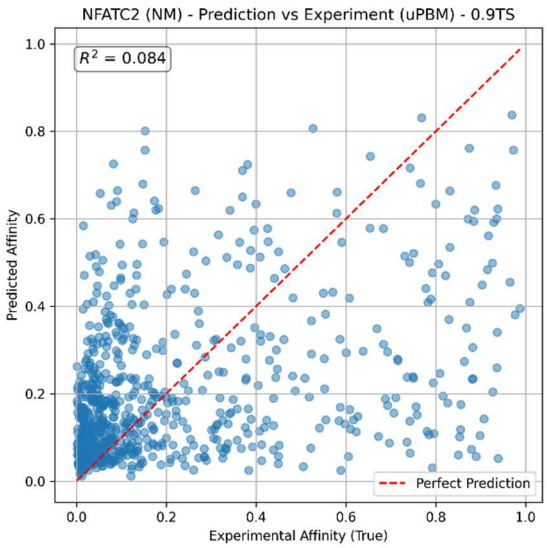


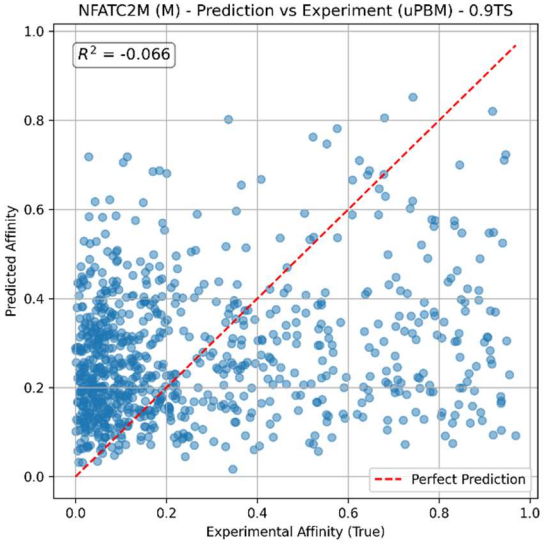

LHX9 (NM) - Prediction vs Experiment (uPBM) - 0.9TS
R^2 = 0.330
Perfect Prediction
Predicted Affinity
Experimental Affinity (True)

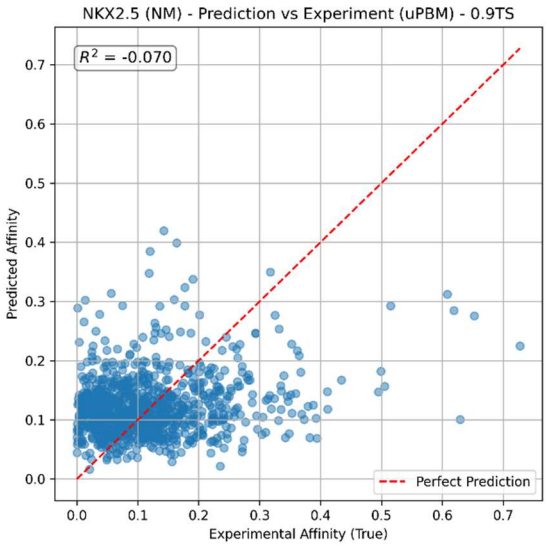

LHX9M (M) - Prediction vs Experiment (uPBM) - 0.9TS
R^2 = 0.299
Perfect Prediction
Predicted Affinity
Experimental Affinity (True)

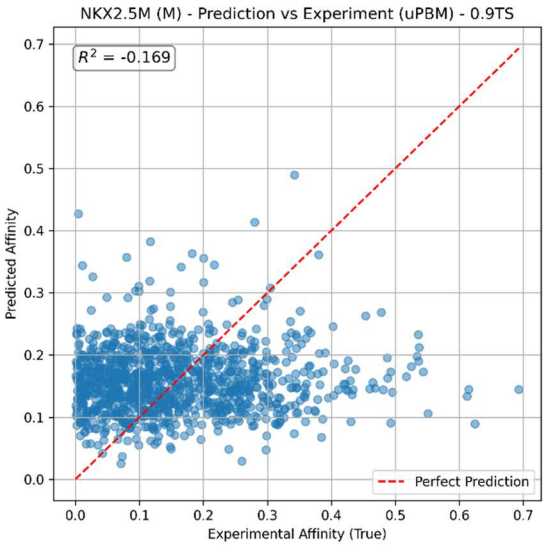

MAX (NM) - Prediction vs Experiment (uPBM) - 0.9TS
R^2 = 0.178
Perfect Prediction
Predicted Affinity
Experimental Affinity (True)

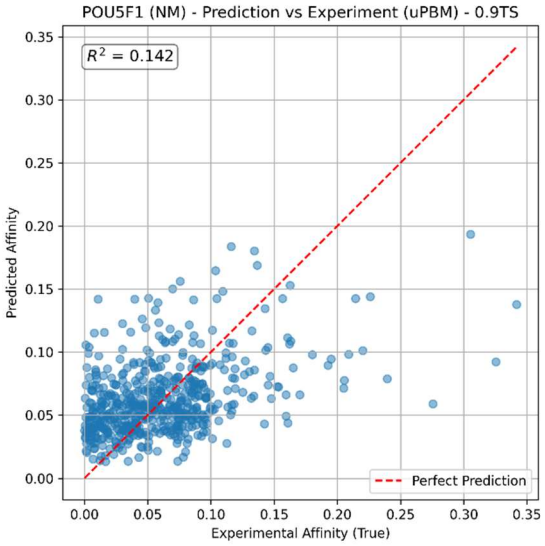

MAXM (M) - Prediction vs Experiment (uPBM) - 0.9TS
R^2 = 0.041
Perfect Prediction
Predicted Affinity
Experimental Affinity (True)

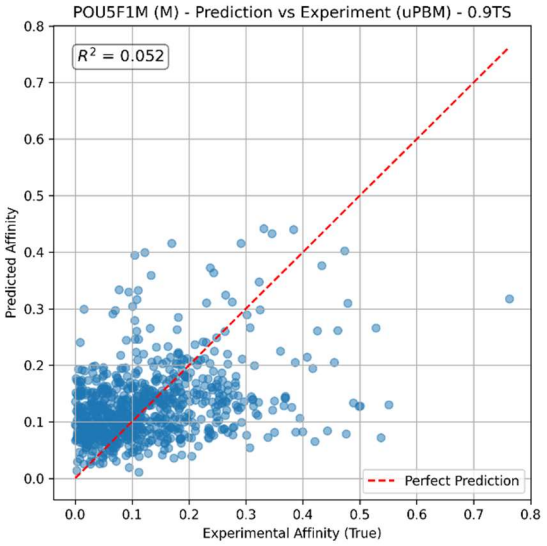

NFATC2 (NM) - Prediction vs Experiment (uPBM) - 0.9TS
R^2 = 0.084
Perfect Prediction
Predicted Affinity
Experimental Affinity (True)


NFATC2M (M) - Prediction vs Experiment (uPBM) - 0.9TS
R^2 = -0.066
Perfect Prediction
Predicted Affinity
Experimental Affinity (True)


NKX2.5 (NM) - Prediction vs Experiment (uPBM) - 0.9TS
R^2 = -0.070
Predicted Affinity
Experimental Affinity (True)
Perfect Prediction


NKX2.5M (M) - Prediction vs Experiment (uPBM) - 0.9TS
R^2 = -0.169
Predicted Affinity
Experimental Affinity (True)
Perfect Prediction

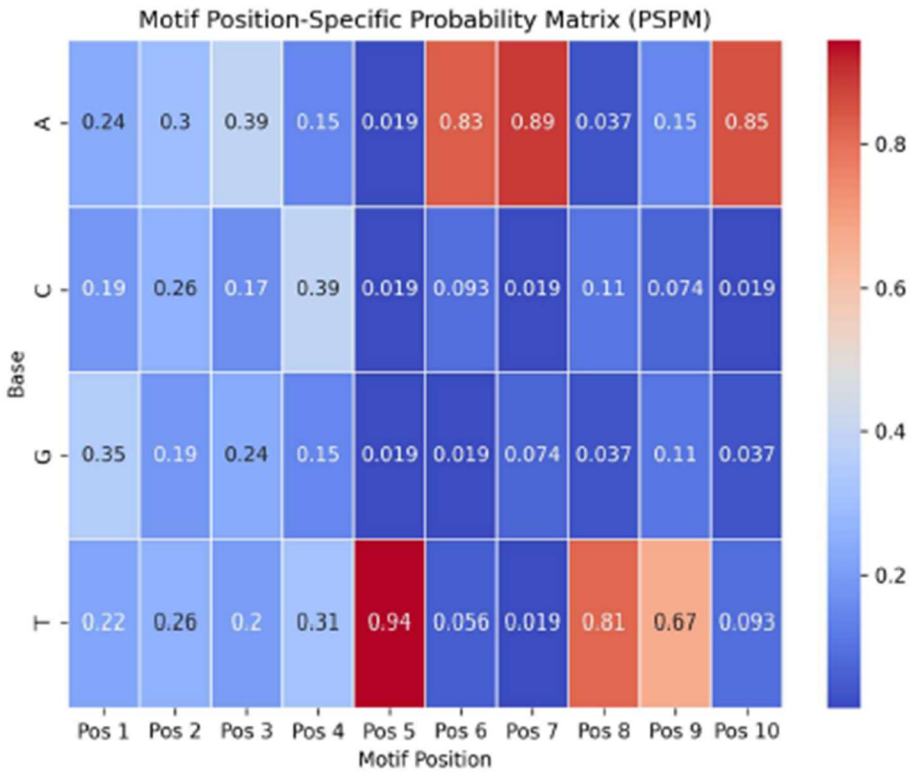

POU5F1 (NM) - Prediction vs Experiment (uPBM) - 0.9TS
R^2 = 0.142
Predicted Affinity
Experimental Affinity (True)
Perfect Prediction


POU5F1M (M) - Prediction vs Experiment (uPBM) - 0.9TS
R^2 = 0.052
Predicted Affinity
Experimental Affinity (True)
Perfect Prediction

ANNEX C: uPBM PSPM, LOGOPLOT AND CONSENSUS (MEME SUITE vs EM ALGORITHM)

LHX9:

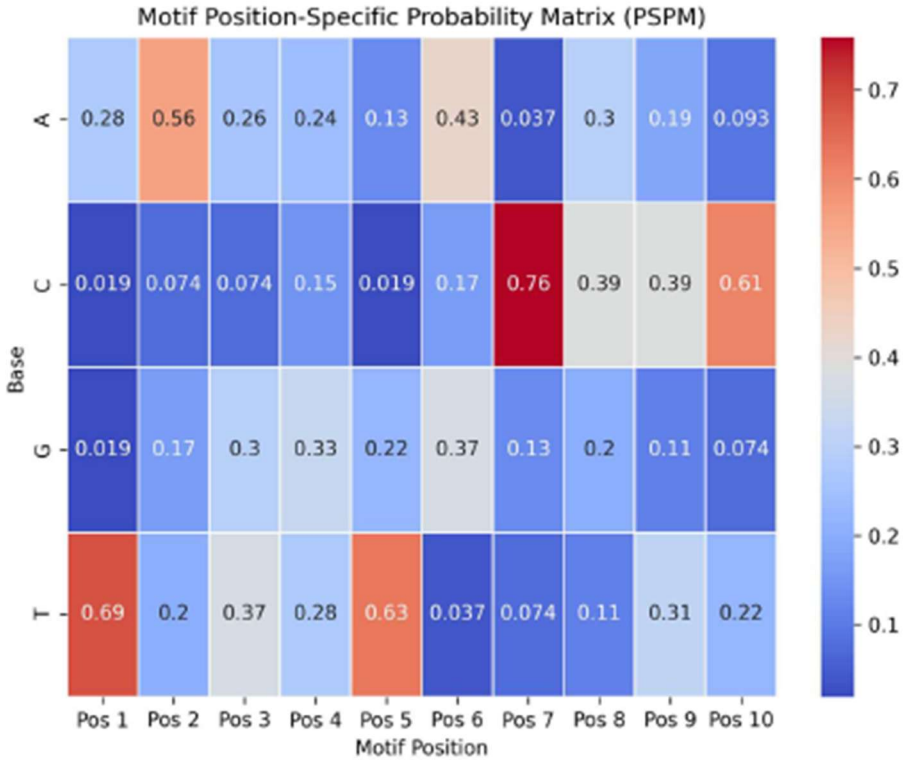
A	0.14	0.00	1.00	1.00	0.00	0.16	0.94	0.08	0.20	0.00
C	0.28	0.00	0.00	0.00	0.16	0.10	0.06	0.16	0.28	0.24
G	0.20	0.00	0.00	0.00	0.00	0.16	0.00	0.46	0.38	0.30
T	0.38	1.00	0.00	0.00	0.84	0.58	0.00	0.30	0.14	0.46



Consensus: GAACTAATTA

NKX2.5

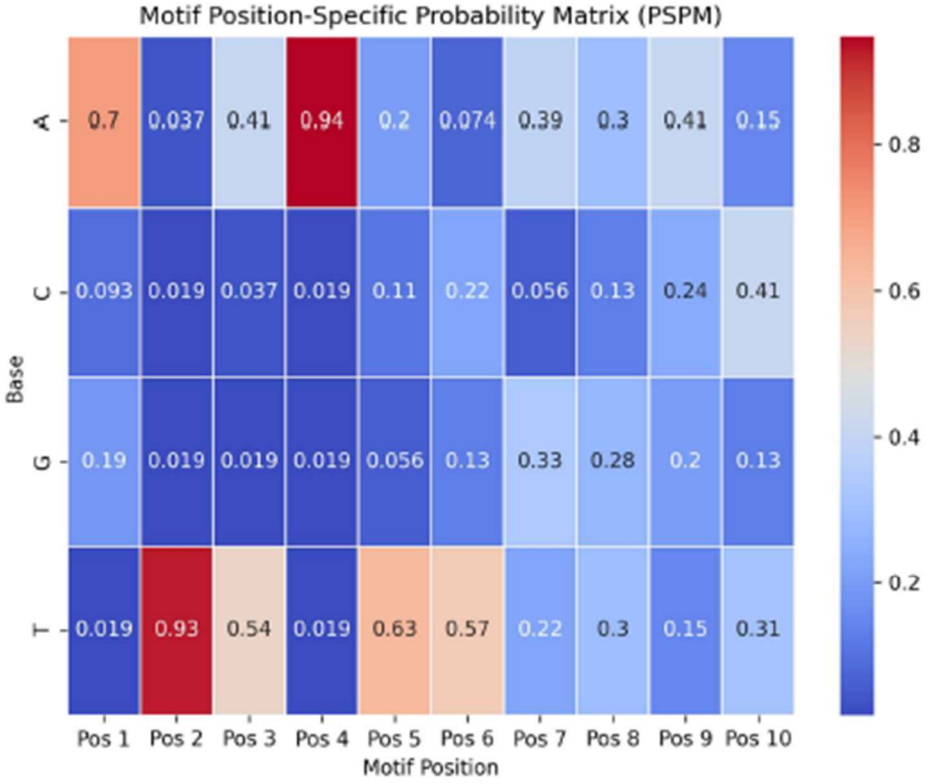
A	0.10	0.15	0.03	0.98	0.95	0.00	0.00	0.18	0.00	0.20
C	0.43	0.18	0.35	0.00	0.05	0.00	0.00	0.00	0.40	0.30
G	0.28	0.03	0.03	0.03	0.00	0.98	0.00	0.83	0.48	0.23
T	0.20	0.65	0.60	0.00	0.00	0.03	1.00	0.00	0.13	0.28



Consensus sequence: GAACTAATTA

DLX3:

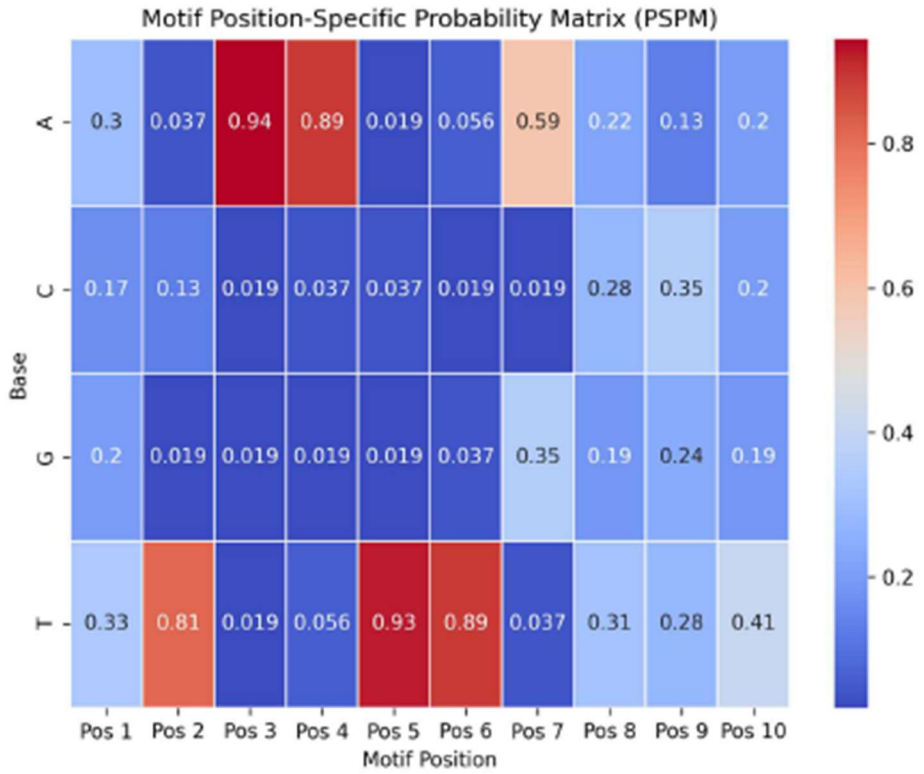
A	0.24	0.40	0.14	0.98	1.00	0.00	0.06	0.56	0.02	0.24
C	0.26	0.18	0.00	0.00	0.00	0.00	0.10	0.00	0.34	0.26
G	0.44	0.10	0.00	0.02	0.00	0.00	0.20	0.44	0.34	0.20
T	0.06	0.32	0.86	0.00	0.00	1.00	0.64	0.00	0.30	0.30



Consensus sequence: ATTATTAAAC

POU5F1:

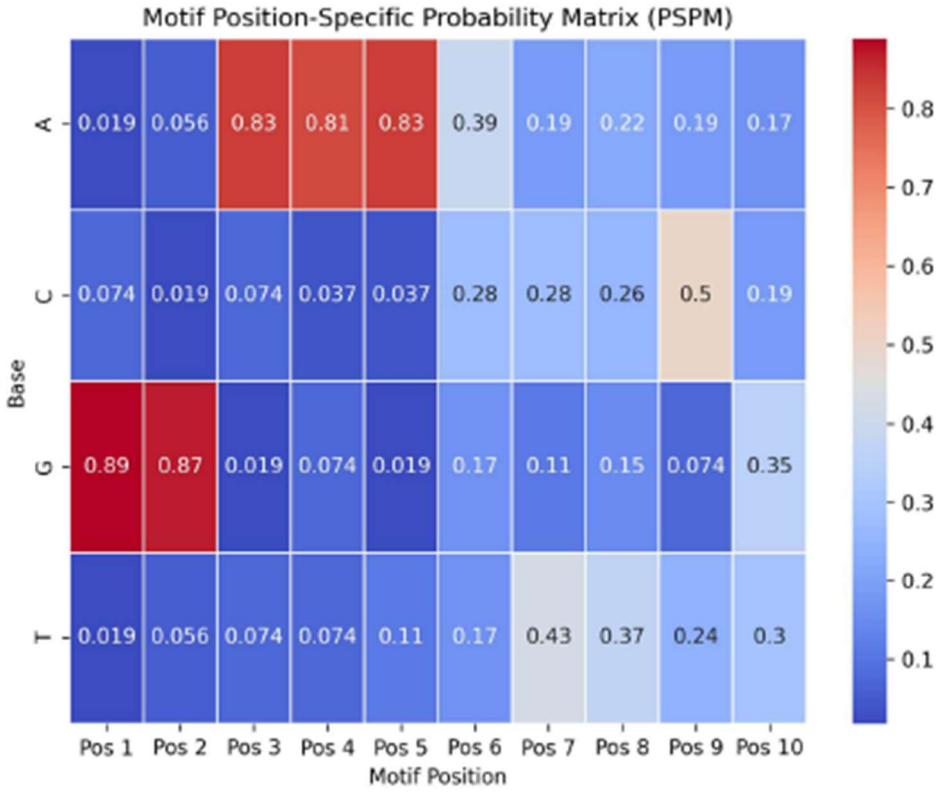
A	0.25	0.35	0.00	1.00	0.96	0.00	0.00	0.52	0.13	0.13
C	0.50	0.15	0.04	0.00	0.00	0.00	0.00	0.00	0.46	0.31
G	0.17	0.02	0.00	0.00	0.00	0.00	0.02	0.44	0.19	0.08
T	0.08	0.48	0.96	0.00	0.04	1.00	0.98	0.04	0.23	0.48



Consensus sequence: TTAATTATCT

NFATC2:

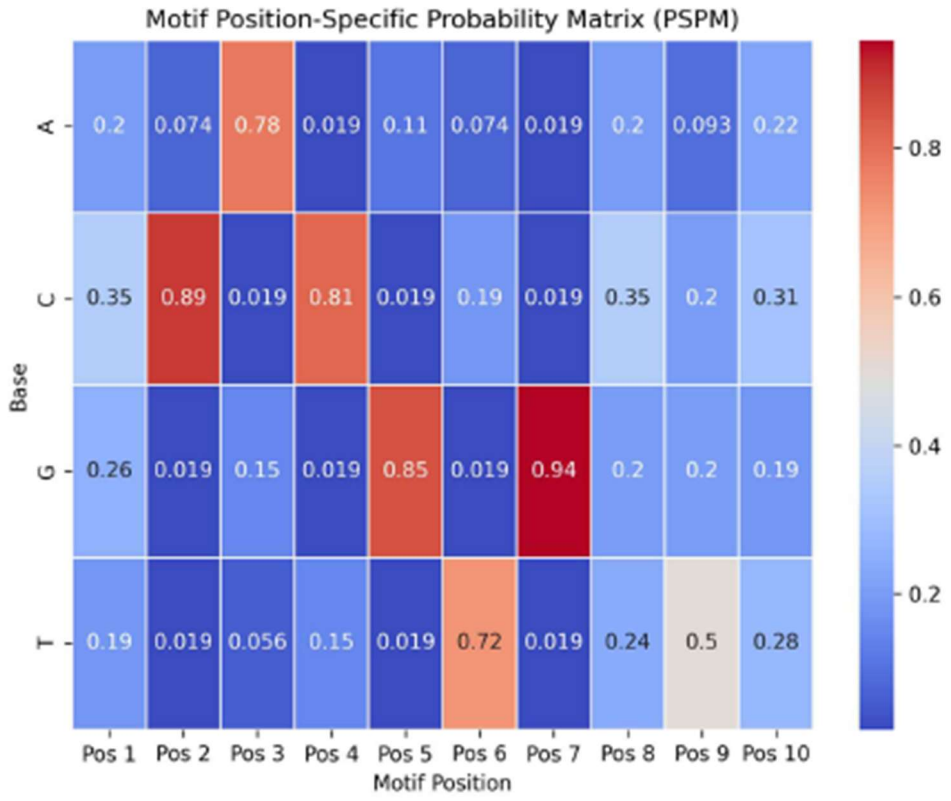
A	0.18	0.48	0.46	0.10	0.00	0.00	1.00	1.00	0.88	0.38
C	0.28	0.24	0.08	0.28	0.00	0.00	0.00	0.00	0.02	0.28
G	0.18	0.14	0.26	0.00	1.00	1.00	0.00	0.00	0.00	0.18
T	0.36	0.14	0.20	0.62	0.00	0.00	0.00	0.00	0.10	0.16



Consensus sequence: GGAAAATTCG

MAX:

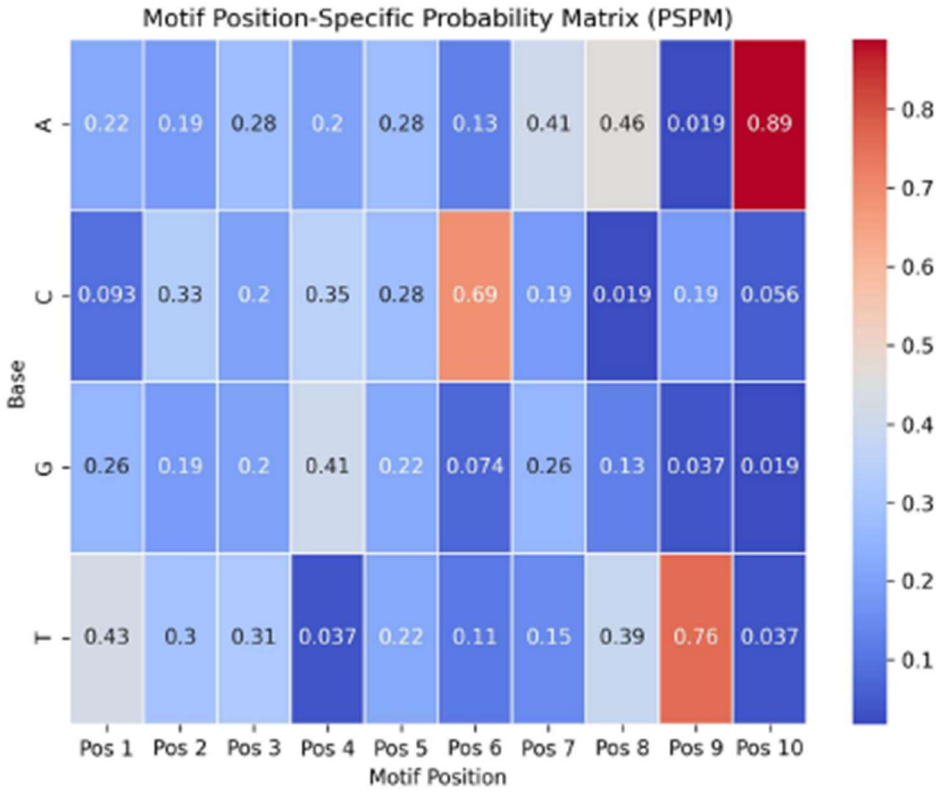
A	0.16	0.00	0.78	0.00	0.10	0.06	0.00	0.30	0.00	0.32
C	0.48	1.00	0.00	0.82	0.00	0.10	0.00	0.36	0.16	0.40
G	0.30	0.00	0.18	0.00	0.90	0.00	0.96	0.12	0.16	0.22
T	0.06	0.00	0.04	0.18	0.00	0.84	0.04	0.22	0.68	0.06



Consensus sequence: CCACGTGCTC

CUX1:

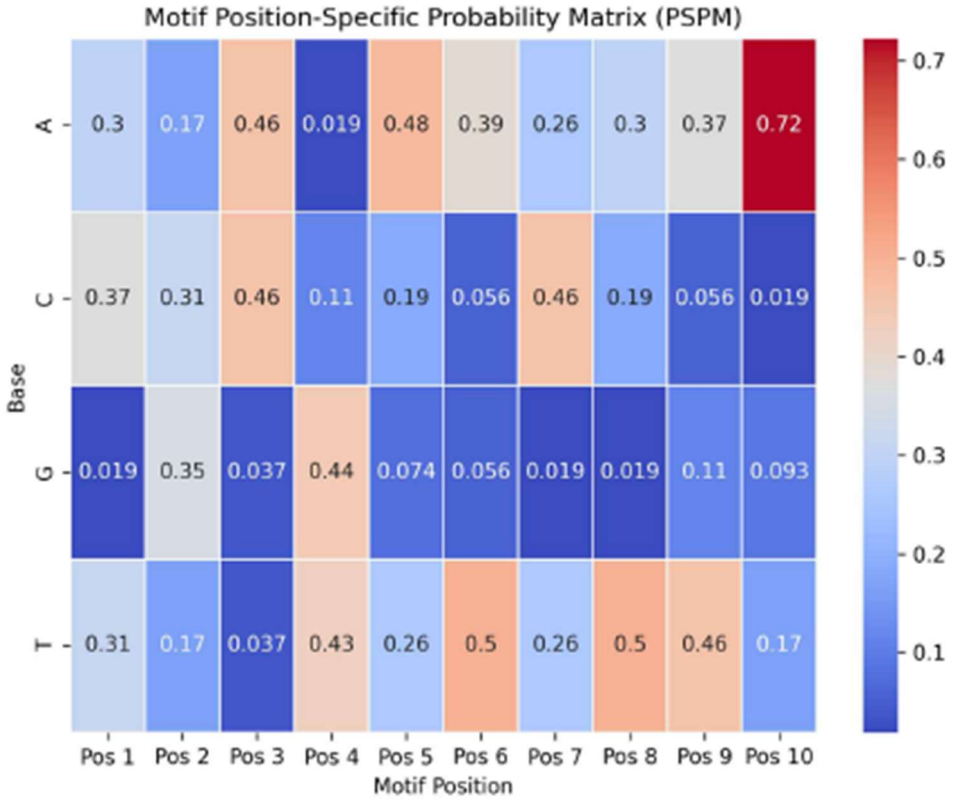
A	0.30	0.44	0.00	0.00	0.42	0.94	0.02	0.86	0.30	0.36
C	0.32	0.06	0.48	1.00	0.00	0.00	0.00	0.00	0.14	0.22
G	0.14	0.28	0.18	0.00	0.54	0.00	0.00	0.06	0.24	0.22
T	0.24	0.22	0.34	0.00	0.04	0.06	0.98	0.08	0.32	0.20



Consensus sequence: TCTGACAATA

CUX2:

A	0.00	0.00	0.35	1.00	0.00	0.63	0.47	0.18	0.06	0.18
C	0.53	0.92	0.02	0.00	0.00	0.27	0.14	0.51	0.39	0.10
G	0.00	0.00	0.63	0.00	0.00	0.02	0.24	0.18	0.22	0.22
T	0.47	0.08	0.00	0.00	1.00	0.08	0.14	0.12	0.33	0.49



Consensus sequence: CGAGATCTTA

ANNEX D: *uPBM_Preprocess & selex.py*

uPBM_Preprocess.py

```
import numpy as np
import pandas as pd
from Bio import pairwise2
import re
from collections import defaultdict
import random
from Bio import motifs
from Bio import SeqIO
import random
import matplotlib.pyplot as plt
import seaborn as sns
import re
from collections import defaultdict
from IPython.display import display
from Bio.Seq import Seq
import sys

# Must change the protein of interest to make the code run

protein = sys.argv[1]


# We first preprocess the raw data file to extract the 35mers and
intensity (we have already deleted the header)

raw_data =
pd.read_csv(f'raw_data/GSM2479934_{protein}_RawData_Cytosine.txt',
sep='\t', skiprows=53)
raw_data = raw_data.dropna(subset='Annotation 2')
raw_data = raw_data.reset_index(drop=True)
raw_data = raw_data[raw_data['Annotation 1'] != 'NoDNA']

# Extract only sequence, mean signal instensity and background
instensity (Annotation 2, Signal Mean, and Background)
raw_data = pd.DataFrame(raw_data[['Annotation 2', 'Signal Mean',
'Background Mean']])
raw_data.columns = ['pbm_sequence', 'mean_signal_intensity',
'mean_background_intensity']

# Trim sequences to 35mers of interest
raw_data['pbm_sequence'] = raw_data['pbm_sequence'].str[:35]
raw_data

"""##### Take the top sequences and generates a FASTA file"""

matrix = raw_data
matrix = matrix.dropna()
matrix = matrix[matrix['pbm_sequence'].str.len() ==
35].reset_index(drop=True)

# Calculate affinities for all sequences
matrix["affinity"] = matrix["mean_signal_intensity"] -
matrix["mean_background_intensity"]
sequences = np.array(matrix[["pbm_sequence"]])
affinities = np.array(matrix[["affinity"]])
max_aff = max(affinities)[0]
```

```
# Determine upper_bound for affinity selection (top 50 highest
affinities)
sort_aff = sorted(list(affinities))
upper_bound = sort_aff[-50][0]
lower_bound = sort_aff[int(len(sort_aff)/4)][0]

# Save top sequences (affinity > upper_bound) to FASTA
with open(f'{protein}_FASTA_seq.txt', 'w') as file:
    for i in np.arange(0, len(sequences)):
        if affinities[i][0] > upper_bound:
            if affinities[i][0] == max_aff:
                file.write(f'>MAX\n{sequences[i][0]}\n')
            else:
                file.write(f'>{i}\n{sequences[i][0]}\n')

import matplotlib.pyplot as plt
plt.plot(sorted(list(affinities)))
# ----------------------
# TOP 50 MOTIF ANALYSIS
# ----------------------
top50 = matrix.sort_values("affinity", ascending=False).head(50)
top_sequences = list(top50["pbm_sequence"])

# Expectation-Maximization
motif_length = 10
num_iterations = 100
random.seed(42)
motif_positions = [random.randint(0, len(seq) - motif_length) for seq
in top_sequences]

def expectation_maximization(sequences, motif_positions, motif_length,
num_iterations):
    bases = ['A', 'C', 'G', 'T']
    for _ in range(num_iterations):
        counts = {base: [1] * motif_length for base in bases}
        for seq, pos in zip(sequences, motif_positions):
            for i in range(motif_length):
                counts[seq[pos + i]][i] += 1
        pspm = {base: [counts[base][i] / sum(counts[b][i] for b in
bases) for i in range(motif_length)] for base in bases}
        new_positions = []
        for seq in sequences:
            best_score = -float("inf")
            best_pos = 0
            for i in range(len(seq) - motif_length + 1):
                score = sum(np.log(pspm[seq[i + j]][j]) for j in
range(motif_length))
                if score > best_score:
                    best_score = score
                    best_pos = i
            new_positions.append(best_pos)
        motif_positions = new_positions
    return pspm, motif_positions

pspm, motif_positions = expectation_maximization(top_sequences,
motif_positions, motif_length, num_iterations)

# ----------------------
# PWM & CONSENSUS
# ----------------------
```

```
df_pspm = pd.DataFrame(pspm, index=[f"Pos {i+1}" for i in
range(motif_length)])
background_freq = {'A': 0.25, 'C': 0.25, 'G': 0.25, 'T': 0.25}
pwm = {base: [np.log2(pspm[base][i] / background_freq[base]) for i in
range(motif_length)] for base in pspm}
df_pwm = pd.DataFrame(pwm, index=[f"Pos {i+1}" for i in
range(motif_length)])
consensus = ''.join(df_pspm.idxmax(axis=1).values)

print("Consensus sequence:", consensus)

# ----------------------
# ALIGN TO CONSENSUS
# ----------------------
def comp(seq):
    return str(Seq(seq).reverse_complement())

def alignment(seq, motif):
    align1 = pairwise2.align.localms(seq, motif, 1, 0, -9999, -9999)
    align2 = pairwise2.align.localms(seq, comp(motif), 1, 0, -9999, -
9999)
    if align1 and align2:
        best = align2[0] if align2[0][2] > align1[0][2] else align1[0]
    elif align1:
        best = align1[0]
    elif align2:
        best = align2[0]
    else:
        return None
    return ''.join([best[0][i] for i in range(len(best[0])) if
best[1][i] != '-'])

aligned_motifs = [alignment(seq, consensus)[:motif_length] for seq in
top_sequences if alignment(seq, consensus)]

print("\nFirst 5 extracted 10-mers aligned to consensus:")
print(aligned_motifs[:5])

# ----------------------
# SAVE RESULTS
# ----------------------
df_pspm.to_csv("motif_pspm.txt", sep="\t", index=True)
df_pwm.to_csv("motif_pwm.txt", sep="\t", index=True)

plt.figure(figsize=(8, 6))
sns.heatmap(df_pspm.T, annot=True, cmap="coolwarm", linewidths=0.5)
plt.title("Motif Position-Specific Probability Matrix (PSPM)")
plt.xlabel("Motif Position")
plt.ylabel("Base")
plt.tight_layout()
plt.savefig("motif_pspm_heatmap.png", dpi=300)
plt.close()

print("\nPWM, PSPM, consensus, aligned motifs, and heatmap saved.")

"""### Now the user needs to go to meme suite to extract the motif
matrix"""

file_path = "motif_pspm.txt"

# Read the file while setting the first column as the index
```

```
df_pspm = pd.read_csv(file_path, sep="\t", index_col=1)

# Remove the first column (if needed)
df_pspm = df_pspm.iloc[:, 1:]  # Removes the first column while
keeping others

# Display the modified matrix
print("Updated PSPM Matrix (First Column Removed):")
print(df_pspm)

# Save the modified version back to a file
df_pspm.to_csv(f'{protein}_freq_matrix.txt', sep="\t", index=True)

print("Updated PSPM saved as 'motif_pspm_modified.txt'")

# reads the frequency matrix and picks the most likely sequence
freq_matrix = pd.read_csv(f'{protein}_freq_matrix.txt',
delim_whitespace=True)
freq_matrix = np.array(freq_matrix)

## generates the consensus (and second sequence for oversampling if
necessary)
cons = []
for i in np.arange(0,len(freq_matrix)):
    result = np.where(freq_matrix[i] == np.amax(freq_matrix[i]))
    cons.append(result[0][0])

translate = {'A':0, 'C':1, 'G':2, 'T':3}
reverse = {0:'A', 1:'C', 2:'G', 3:'T'}

seq = ''
for num in cons:
    seq = seq+reverse[num]

consensus = [seq]

'''
# second
sec = []
for i in np.arange(0,len(freq_matrix)):
    if np.amax(freq_matrix[i])<0.5:
        second = sorted(freq_matrix[i])[-2]
        result = list(freq_matrix[i]).index(second)
        sec.append(result)
        continue
    result = np.where(freq_matrix[i] == np.amax(freq_matrix[i]))
    sec.append(result[0][0])

seq = ''
for num in sec:
    seq = seq+reverse[num]

consensus.append(seq)
'''
consensus

max_aff = max(affinities)[0]
min_aff = min(affinities)[0]
normalization = max_aff-min_aff

# define a new function: best alignment between the consensus and a
```

```
sequence from the 35mers
sequences=np.array(ams[["pbm_sequence"]])

from Bio import pairwise2
import re

def comp(seq):
    """Returns the reverse complement of a DNA sequence."""
    complement = str.maketrans("ACGTacgt", "TGCAtgca")
    return seq.translate(complement)[::-1]

def alignment(seq, aln):
    """Aligns a sequence to another and removes gaps from the best
alignment."""

    # Get alignments for both original and reverse-complement
    align1 = pairwise2.align.localms(seq, aln, 1, 0, -9999, -9999)
    align2 = pairwise2.align.localms(seq, comp(aln), 1, 0, -9999, -
9999)

    # Ensure alignment is not empty before accessing [0]
    if align1 and align2:
        if align2[0][2] > align1[0][2]:  # Compare alignment scores
            fst, snd, scx, *q = align2[0]
        else:
            fst, snd, scx, *q = align1[0]
    elif align1:  # Use align1 if align2 is empty
        fst, snd, scx, *q = align1[0]
    elif align2:  # Use align2 if align1 is empty
        fst, snd, scx, *q = align2[0]
    else:
        return ""  # No valid alignment found

    # Remove gaps from the aligned sequence
    aligned = ''.join([fst[n] for n, i in enumerate(fst) if snd[n] !=
'-'])
    aligned = re.sub('-', '', aligned)

    return aligned


# generates the file matching the sequences with the consensus
sequence(s)

len_aln = len(consensus[0])

fst_aln = consensus[0]

to_write = []
for seq, aff in zip(sequences, affinities):
    s_1 = alignment(seq[0],fst_aln)
    if len(s_1) != len_aln:
        continue
    if aff[0]<lower_bound:
        continue
    score = pairwise2.align.localms(s_1, fst_aln, 1, 0, -9999, -
9999)[0][2]
    if pairwise2.align.localms(s_1, comp(fst_aln), 1, 0, -9999, -
9999)[0][2] > pairwise2.align.localms(s_1, fst_aln, 1, 0, -9999, -
9999)[0][2]:
        score = pairwise2.align.localms(s_1, comp(fst_aln), 1, 0, -
```

```
9999, -9999)[0][2]
        s_1 = comp(s_1)
    if score<len_aln-3:
        w=1
    if score>=len_aln-3:
        w=10
    to_write.append((s_1, (aff[0]-min_aff)/normalization, w))

if len(consensus) != 1:
    snd_aln = consensus[1]
    for seq, aff in zip(sequences, affinities):
        s_2 = alignment(seq[0],snd_aln)
        if len(s_2) != len_aln:
            continue
        if aff<upper_bound:
            continue
        a = aff[0]-0.0001
        score = pairwise2.align.localms(s_2, snd_aln, 1, 0, -9999, -
9999)[0][2]
        if pairwise2.align.localms(s_2, comp(snd_aln), 1, 0, -9999, -
9999)[0][2] > pairwise2.align.localms(s_2, snd_aln, 1, 0, -9999, -
9999)[0][2]:
            score = pairwise2.align.localms(s_2, comp(snd_aln), 1, 0,
-9999, -9999)[0][2]
            s_2 = comp(s_2)
        if score<len_aln-3:
            w=1
        if score>=len_aln-3:
            w=10
        to_write.append((s_2, aff[0], w))


with open(f'{protein}_alignment_weighted.txt','w') as file:
    file.write('ID_REF\tVALUE\tWEIGHT\n')

with open(f'{protein}_alignment_weighted.txt','a') as file:
    for i in to_write:
        file.write(f'{i[0]}\t{i[1]}\t{i[2]}\n')

# from the previous file (the file of nicely aligned kmers), we remove
those that have low score according to the freq matrix
data = pd.read_csv(f'{protein}_alignment_weighted.txt', sep='\t')
dodecamers = np.array(data[["ID_REF"]])
affs = np.array(data[["VALUE"]])
ws = np.array(data[["WEIGHT"]])

freq_matrix = pd.read_csv(f'{protein}_freq_matrix.txt',
delim_whitespace=True)
freq_matrix = np.array(freq_matrix)

# scores
scores = []
for dod in dodecamers:
    score = 0
    for j in np.arange(0,len_aln):
        a = translate[dod[0][j]]
        score = score + freq_matrix[j][a]
    scores.append(score)

# boundaries
low_score = len_aln/4
```

```
upp_score = sorted(scores)[::-1][50]

# file
to_write = []

for dod, aff, w, score in zip(dodecamers, affs, ws, scores):
    if score<low_score:
        continue
    if score<upp_score:
        #to_write.append((dod[0],aff[0], w[0], score, 1))
        to_write.append((dod[0],aff[0], 1))
    if score>upp_score:
#        to_write.append((dod[0],aff[0], w[0], score, 10))
        to_write.append((dod[0],aff[0], 10))
with open(f'{protein}_matrix_aligned.txt','w') as file:
#    file.write('ID_REF\tVALUE\tWEIGHT\tSCORE\tWEIGHTEDSCORE\n')
    file.write('ID_REF\tVALUE\tWEIGHT\n')

with open(f'{protein}_matrix_aligned.txt','a') as file:
    for i in to_write:
#        file.write(f'{i[0]}\t{i[1]}\t{i[2]}\t{i[3]}\t{i[4]}\n')
        file.write(f'{i[0]}\t{i[1]}\t{i[2]}\n')

deca = pd.read_csv(f'{protein}_matrix_aligned.txt', sep='\t')

strings = list(deca["ID_REF"])
values = list(deca["VALUE"])
weights = list(deca["WEIGHT"])  # pick either WEIGHT or WEIGHTEDSCORE

values = [(value-min(values))/(max(values)-min(values)) for value in
values]

dictionary = dict(zip(values, strings))
weight_dict = dict(zip(values, weights))

len_aln = len(strings[0])

"""#### Last step, we undersample"""

#### undersampling

# number of divisions we are using
L = 15000

# affinitiy boundaries
sort_aff = sorted(values)
# pick top points ~ proportion 1:500 before undersampling i.e. we pick
50 top points not to undersample
upper_bound = sort_aff[-1000]

# interval dictionary -- maybe it is not the best option? idk it works
for now...
interval_dict = defaultdict(list)
for value in values:
    interval_dict[int(value*L)].append(value)

# creates a matrix of values: we pick one value for the intervals
where i<L/2
# and up to two for i>L/2

training_ordered = []
```

```
start = 1 # i.e. starts picking values larger than start/L from the
normalized affinites

for i in np.arange(start,L+1):
    if len(interval_dict[i])>0:
        if i <= L*upper_bound:
            random_values = [random.choice(interval_dict[i]) for j in
np.arange(0,20)]
            random_values = set(random_values)
            value = random.choice(interval_dict[i])
            training_ordered.append([dictionary[value], value,
weight_dict[value]])
        if i > L*upper_bound:
            for value in interval_dict[i]:
                training_ordered.append([dictionary[value], value,
weight_dict[value]])

# # writes the file w/o randomization (not used but in case we need
it)

# with open(f'{protein}_training_ordered.txt','w') as file:
#     file.write('ID_REF\tVALUE\tWEIGHT\n')

# with open(f'{protein}_training_ordered.txt','a') as file:
#     for vector in training_ordered:
#         file.write("%s\t" % vector[0])
#         file.write("%s\t" % vector[1])
#         file.write("%s\n" % vector[2])

# writes the randomized file ready for training

np.random.shuffle(training_ordered)

with open(f'{protein}_training.txt','w') as file:
    file.write('ID_REF\tVALUE\tWEIGHT\n')

with open(f'{protein}_training.txt','a') as file:
    for vector in training_ordered:
        file.write("%s\t" % vector[0])
        file.write("%s\t" % vector[1])
        file.write("%s\n" % vector[2])



# Convert full values and undersampled values to arrays for plotting
full_affinities = np.array(values)

undersampled_affinities = [val for _, val, _ in training_ordered]

# Plot histogram of affinities before and after undersampling
plt.figure(figsize=(12, 5))

# --- Before undersampling ---
plt.subplot(1, 2, 1)
sns.histplot(full_affinities, bins=100, color='skyblue', kde=False)
plt.axvline(x=upper_bound, color='red', linestyle='--', label=f'Upper
bound ({upper_bound:.3f})')
plt.title("Before Undersampling")
plt.xlabel("Normalized Affinity")
plt.ylabel("Count")
plt.legend()
```

```
# --- After undersampling ---
plt.subplot(1, 2, 2)
sns.histplot(undersampled_affinities, bins=100,
color='mediumseagreen', kde=False)
plt.axvline(x=upper_bound, color='red', linestyle='--', label=f'Upper
bound ({upper_bound:.3f})')
plt.title("After Undersampling")
plt.xlabel("Normalized Affinity")
plt.ylabel("Count")
plt.legend()

plt.tight_layout()
plt.show()
```

selex.py

```
import pandas as pd
from Bio import SeqIO
from collections import defaultdict
import random
import math
import os
import shutil
import sys

def load_sequences(filepath, center_length=20):
    """Extract central region from 40-mer, excluding sequences with
'N'."""
    sequences = []
    for record in SeqIO.parse(filepath, "fastq"):
        seq = str(record.seq)
        if len(seq) == 40:
            start = (40 - center_length) // 2
            center_seq = seq[start:start + center_length]
            if 'N' not in center_seq:
                sequences.append(center_seq)
    return sequences

def split_train_test(sequences, test_fraction=0.3):
    random.shuffle(sequences)
    cutoff = int(len(sequences) * (1 - test_fraction))
    return sequences[:cutoff], sequences[cutoff:]

def build_markov_model(sequences, order=3):
    model = defaultdict(lambda: defaultdict(int))
    context_counts = defaultdict(int)
    for seq in sequences:
        for i in range(len(seq) - order):
            context = seq[i:i+order]
            next_base = seq[i+order]
            model[context][next_base] += 1
            context_counts[context] += 1
    probs = {}
    for context in model:
        probs[context] = {}
        total = context_counts[context]
        for base in model[context]:
            probs[context][base] = model[context][base] / total
    return probs
```

```
def compute_kmer_stats(sequences, k, markov_model, order):
    kmer_counts = defaultdict(int)
    for seq in sequences:
        for i in range(len(seq) - k + 1):
            kmer = seq[i:i+k]
            kmer_counts[kmer] += 1

    total_kmers = sum(kmer_counts.values())
    kmer_data = []
    for idx, (kmer, count) in enumerate(sorted(kmer_counts.items(),
key=lambda x: -x[1])):
        prob = 1.0
        for i in range(len(kmer) - order):
            context = kmer[i:i+order]
            next_base = kmer[i+order]
            if context in markov_model and next_base in
markov_model[context]:
                prob *= markov_model[context][next_base]
            else:
                prob *= 1e-6  # pseudocount smoothing
        expected = total_kmers * prob
        affinity = count / expected if expected > 0 else 0
        se = (affinity / math.sqrt(count)) if count > 0 and expected >
0 else 0
        kmer_data.append([idx, kmer, count, prob, expected, affinity,
se])
    df = pd.DataFrame(kmer_data, columns=["#", "Kmer",
"ObservedCount", "Probability", "ExpectedCount", "Affinity", "SE"])
    df = df.sort_values(by="Affinity",
ascending=False).reset_index(drop=True)
    df["#"] = df.index
    return df

# === Main Execution ===

base_dir = f'fastq{sys.argv[2]}_folder/{sys.argv[1]}'
output_dir = os.path.join(base_dir, "six3")
result_dir = os.path.join(base_dir, "RESULT")
os.makedirs(output_dir, exist_ok=True)
os.makedirs(result_dir, exist_ok=True)

rounds = ["R0", "R1", "R2", "R3"]
files = {r: os.path.join(base_dir, f"{r}.fastq") for r in rounds}

# Load R0 and build model
r0_seqs = load_sequences(files["R0"])
r0_train, r0_test = split_train_test(r0_seqs)
markov_model = build_markov_model(r0_train, order=3)

# Analyze R1-R3
for r in rounds[1:]:
    seqs = load_sequences(files[r])
    df = compute_kmer_stats(seqs, k=10, markov_model=markov_model,
order=3)
    output_file = os.path.join(output_dir, f"six3_0vs{r[1]}.txt")
    df.to_csv(output_file, sep='\t', index=False)
    print(f"Written: {output_file} with {len(df)} k-mers.")

    # Save copy of R3 as 3.txt in RESULT
    if r == "R3":
```

```
        result_copy = os.path.join(result_dir, "3.txt")
        shutil.copyfile(output_file, result_copy)
        print(f"Copied to: {result_copy}")
```

To consult the regressor itself or its features, please look at the DNAffinity repository on GitHub: https://github.com/Jalbiti/DNAffinity .