McGill University

Computational Biology and Research

COMP 561

Prediction of GATA-1 Transcription Factor Binding Based on DNA Physical Properties

Author
Yasmin Salehi

Personal Declaration

I wish to state that the work presented here has been carried out by the author alone. No part of it has been copied from the internet or other sources, unless clearly acknowledged.

Dec, 14, 2018

Contents

1 Introduction

Recognition of specific DNA sequences by proteins is essential in regulation of gene expression. Transcription, a process in which mRNA is created from a DNA template, is initiated after a set of transcription factors (TF) assemble on specific regions of the DNA, to facilitate RNA polymerase locate and bind to DNA promoter regions. This mechanism is governed by direct readout of the amino acid sequences as well as indirect readout of the DNA's conformation or deformability, which are often referred to as "sequence readout" and "shape readout" respectively [5][9]. This knowledge enables establishment of models which can predict TF protein binding sites, that are applicable to "predicting effects of mutation on gene regulation, and elucidating the differences between related transcription factors"[2].

Current experimental methods for finding TF binding sites combines ChIP-Seq with a computational scan for the TF's position weight matrix to identify sites along the DNA which are occupied by different TFs [7]. Additionally, there are available techniques which allow for finding protein positive binding sites, where a transcription factor may or may not bind to. The goal of this project is to propose a black box approach where an appropriate machine learning classifier is trained with multiple features to distinguish between bound and non-bound regions of the DNA for a specific TF based on DNA's physical properties. The most important physical property with which the ML classifier will be trained with is the minor groove width. Variations in minor groove width play a key role in determining shape complementarity between the DNA ligand and protein surface [5][8]. Roll, propeller twist and helical twist are other important physical properties, which determine protein binding affinity to positive binding regions [7]. To keep the project manageable and executable, the selected ML classifier will be trained such that it can predict GATA1 bound and non-bound sites alongside Chromosome 1 only.

2 Methodology

In order to distinguish the regions that are bound to GATA1 from those that are not based on the predicted structural properties of the DNA, one needs to identify a set of bound and non-bound DNA sequences by GATA1, calculate their physical properties, and use the subsequent data to train the ML classifier. Herein, the following subsections explain the approach for each of these steps in more detail.

2.1 Distinguishing Non-Bound Regions from Positive Regions

To distinguish the active regulatory sites alongside chromosome 1, the data in Reference [3] needs to be processed to extract rows in which their 1^{st}column value is equal to "chr1". In Python, this task can be easily done by loading the text file contents to Python Pandas, and writing the data to a new *csv* file only when the 1^{st} column value is equal to "chr1". The same approach can be exploited to extract the GATA1 binding sites in chromosome 1 from other TF binding sites by processing the data in Reference [4], to filter out rows in which their 2^{nd} and 5^{th} column values are set to "chr1" and "GATA1". While writing the later output file, the 1^{st}, 6^{th}, and the 7^{th} columns may be dropped. Table 1 and Table 2 are indicative of the formatting of the output files, which are of type *.bed*, representing the positive and bound regions.

Chrom	ChromStart	ChromEnd
chr1	237550	237989
chr1	521310	521756
chr1	713702	714675
chr1	762022	763345
chr1	805100	805473
chr1	839802	841023

Table 1: A few examples of positive regions

Chrom	ChromStart	ChromEnd
chr1	840970	840980
chr1	959728	959738
chr1	1045283	1045293

Table 2: A few examples of bound regions

To differentiate the bound from the non-bound regions, Algorithm 1 was devised in Python, which takes 4 arrays as inputs that are representative of the starting and ending chromosome indices in the bound and positive regions, namely: bound_start[], bound_end[], positive_start[], and positive_end[]. This algorithm obeys the *Half-Open UCSC Genome Coordinate Counting System.*

Algorithm 1 Filtering Non-Bound from Positive Regions

```
function subtraction(bound_start, bound_end, positive_start, positive_end)
    length ← length(bound_start)                    ▷ length of the bound_start array
    i ← 0                                                          ▷ bound_start index
    j ← 0                                                            ▷ bound_end index
    k ← 0                                                       ▷ positive_start index
    h ← 0                                                         ▷ positive_end index
    non_bound_start ← []
    non_bound_end ← []
    while length(positive_start) > 0 do
        if bound_start[i] > positive_start[k] and bound_end[j] < positive_end[h] then
            add positive_start[k] to non_bound_start
            add bound_start[i] to non_bound_end
            pop the first element in positive_start
            insert bound_end[j] to the 0th position of positive_start
            if i + 1 < length then
                i ← i + 1
                j ← j + 1
        else if bound_start[i] == positive_start[k] and bound_end[j] < positive_end[h]
    then
            pop the first element in positive_start
            insert bound_end[j] to the 0th position of positive_start
            if i + 1 < length then
```

```
                $i \leftarrow i + 1$
                $j \leftarrow j + 1$
        else if $bound\_start[i] > positive\_start[k]$ and $bound\_end[j] == positive\_end[h]$ then
            add $positive\_start[k]$ to $non\_bound\_start$
            add $bound\_start[i]$ to $non\_bound\_end$
            pop the first element in $positive\_start$
            if $i + 1 < length$ then
                $i \leftarrow i + 1$
                $j \leftarrow j + 1$
        else if $bound\_start[i] > positive\_start[k]$ and $bound\_end[j] > positive\_end[h]$ then
            add $positive\_start[k]$ to $non\_bound\_start$
            add $positive\_end[h]$ to $non\_bound\_end$
            pop the first element in $positive\_start$
            $h \leftarrow h + 1$
        else if $bound\_start[i] == positive\_start[k]$ and $bound\_end[j] == positive\_end[h]$ then
            if $i + 1 < length$ then
                $i \leftarrow i + 1$
                $j \leftarrow j + 1$
        else
            add $positive\_start[k]$ to $non\_bound\_start$
            add $positive\_end[h]$ to $non\_bound\_end$
            pop the first element in $positive\_start$
            if $j <$ length of $positive\_end$ then
                $h \leftarrow h + 1$
            break
    if $h <$ length of $positive\_end$ then
        while length of $positive\_start > 0$ do
            add $positive\_start[0]$ to $non\_bound\_start$
            add $positive\_end[h]$ to $non\_bound\_end$
            pop the first element in $positive\_start$
            $h \leftarrow h + 1$
    return $non\_bound\_start$ and $non\_bound\_end$
```

Running this algorithm on the input files resulted in an output which a small part of it has been depicted in Table 3. Again, the formatting of this output file is of type *.bed*. Outputting files in this format allows for easier manipulation of Reference [6] for retrieving the structural properties of the protein sequences, as it accepts input files of type *.bed*

Lastly, another version of these output files were generated, which included two additional attributes per data point, namely a label that is set to *True* for the bound regions and *False* for the non-bound regions, as well as the protein sequence that has been represented by the starting and ending coordinates. To get the protein sequence for each set of starting and ending coordinates, Reference [1], which includes the sequence of the human genome, alongside a program written in Python which uses the parsing capabilities of the BioPhython package were used, to parse through the *chr1.fa* fasta file and output the nucleotides with

Chrom	ChromStart	ChromEnd
chr1	237550	237989
chr1	521310	521756
chr1	713702	714675
chr1	762022	763345
chr1	805100	805473
chr1	839802	840970
chr1	840980	841023

Table 3: A few examples of non-bound regions

respect to the starting and the ending coordinates. This task was implemented by a method which uses a for loop to output the nucleotides with respect to the starting and the ending coordinates, and merges all the nucleotides in form of one string at the end. These information were later used to train the machine learning classifier.

2.2 Retrieving and Processing Physical Properties Data

As mentioned in the previous section, the bound and non-bound bed files need to be inputted in Reference [6] to retrieve their corresponding physical properties, inclusive of minor groove width (MGW), propeller twist (ProT), helical twist (HelT), and roll (Roll). To output the information in form of one text file per physical attribute, the filter was set to output the maximum number of lines, which was (10,000,000).

To use the data retrieved from the previous section, the following algorithms were devised to parse the output files, and generate relevant data to train the machine learning classifier. To begin, Algorithm 2 was created to "clean" the outputted files from the website, such that they could be parsed as a text file with three columns. This algorithm removes lines which start with words "element" and "variableStep". Afterwards, the starting and the ending chromosome coordinates of the bound and non-bound regions, stored in two separate arrays, were inputted to function *attribute*, depicted in Algorithm 3. This algorithm stores the physical property (either MGW, ProT, HelT, or Roll) of each nucleotide in the protein sequence in a different column within the csv file. Knowing that GATA1 binds to regions of length 10, the program was designed to store 10 physical attributes corresponding to each of the nucleotides within a protein sequence of length 10. Algorithm 4 is representative of how the parser was programmed for bound regions. Algorithm 5 exhibits additional code which needs to be added to Algorithm 4 to parse the non-bound regions. This was because the length of non-bound regions would vary from 10, and to not lose data, one needs to determine all the starting and ending chromosome positions that may occur for sites that are longer than 10. As depicted, this task was done by implementing a for loop which creates additional starting and ending chromosome coordinates if the difference value is greater than 10, and increments the starting position by one at the end of the while loop to account for all motifs of length 10 within the positive binding site. All of the determined starting and ending positions were later stored into two new arrays, namely, *new_chrom_start* and *new_chrom_end*, which were inputted to the function *attribute*. Lastly, when calling the *parser* function, *attribute_name* was either of "MGW", "ProT", "HelT", or "Roll", input was the text file containing either the bound or the non-bound region (in *.bed* format), output was the file outputted from website which corresponded to either of the physical

properties, and *csvFile* was the name of the csv file where all the attributes were saved to be used for later.

Algorithm 2 Cleaning the Output Files

function *removing_lines_with_words*(filename)
Require: filename is open is reading mode
    Read the lines
Ensure: filename is closed
Require: filename is open is writing mode
    for line is lines **do**
        strip and split the line contents by white space
        if the first element was not "variableStep" or "track" **then**
            write the line in filename.
Ensure: filename is closed

Algorithm 3 Attribute Value Setter

function *attribute_outputter*(chromStart, chromStop, filename, attribute_name, csv-file)
    $attribute\_name1 \leftarrow attribute\_name +$ '1'
    $attribute\_name2 \leftarrow attribute\_name +$ '2'
    $attribute\_name3 \leftarrow attribute\_name +$ '3'
    $attribute\_name4 \leftarrow attribute\_name +$ '4'
    $attribute\_name5 \leftarrow attribute\_name +$ '5'
    $attribute\_name6 \leftarrow attribute\_name +$ '6'
    $attribute\_name7 \leftarrow attribute\_name +$ '7'
    $attribute\_name8 \leftarrow attribute\_name +$ '8'
    $attribute\_name9 \leftarrow attribute\_name +$ '9'
    $attribute\_name10 \leftarrow attribute\_name +$ '10'
    $m \leftarrow 0$
    $n \leftarrow 0$
    $att\_array \leftarrow []$ ▷ att_array is a numpy array
Require: filename is open
    Read the lines
    for *chrom* in *ChromStart* **do**
        for *line* in *lines* **do**
            strip and split the *line* contents by white space
            $x \leftarrow line$
            if $x[1] == chrom$ **then**
                add $x[3]$ to att_array
            else if $x[2] == chromStop[n] - 1$ **then**
                add $x[3]$ to att_array
                Write $attribute\_name1$ to $attribute\_name10$ in the csvfile
                empty the att_array
                $m \leftarrow m + 1$
                $n \leftarrow n + 1$
                break
            else if $x[1] > chrom$ and $x[1] < chromeStop[n] - 1$ **then**
                add $x[3]$ to att_array

Ensure: filename is closed

Algorithm 4 Parser for Bound Region Outputs

```
function parser(attribute_name, input, output, csvFile)
    removing_lines_with_words (output)
    chromStart ← []
    chromStop ← []
Require: input is open
    read lines
    for line in lines do
        strip and split the lines by white space
        start ← 1st element in the line
        end ← 2nd element in the line
        add start to chromStart
        add end to chromStop
Ensure: input is closed
    attribute (chromStart, chromStop, output, attribute_name, csvfile)
```

Algorithm 5 Additional code added to *Parser* for non-bound regions

```
function parser(attribute_name, input, output, csvFile)
    new_chromStart ← []
    new_chromStop ← []
    for i < length(chromStart) do
        m ← chromStart[i]
        diff ← chromStop[i] − m
        if diff == 10 then
            add chromStart[i] to new_chromStart
            add chromStop[i] to new_chromStop
        else
            while diff > 10 do
                add m to new_chromStart
                add m + 10 to new_chromStop
                m ← m + 1
                diff = chrom_stop[i] − m
    attribute (new_chromStart, new_chromStop, output, attribute_name, csvfile)
```

When executing the code to get data per each of the physical properties, the program run-time may become very large for non bound regions because of breaking down the non-bound sites to motifs of length 10. Consequently, to reduce the program run-time, it is necessary to sample the data. To ensure that the sampled data represents the data set well and is not biased, the difference values between the starting and ending coordinates of the non-bound regions versus the number of times they occur were plotted as shown in Figure 1. According to this figure, the difference value between 465 to 485 has the highest frequency, and according to Figure 2, this difference value is almost always observed alongside Chromosome 1. Therefore, Algorithm 5 was slightly modified to Algorithm 6, which filters difference values between 390 to 426 as an example. The starting and ending

values determining the range of the for loop in line 11 of Algorithm 6 were changed to values: 100 to 150, 390 to 426, 450 to 455, 480 to 485, 755 to 760, and 1000 to 1050 to include as many data points as possible without biasing the data set and having to deal with the long program run-time. These steps were repeated for all the 4 physical properties before proceeding to the next step.

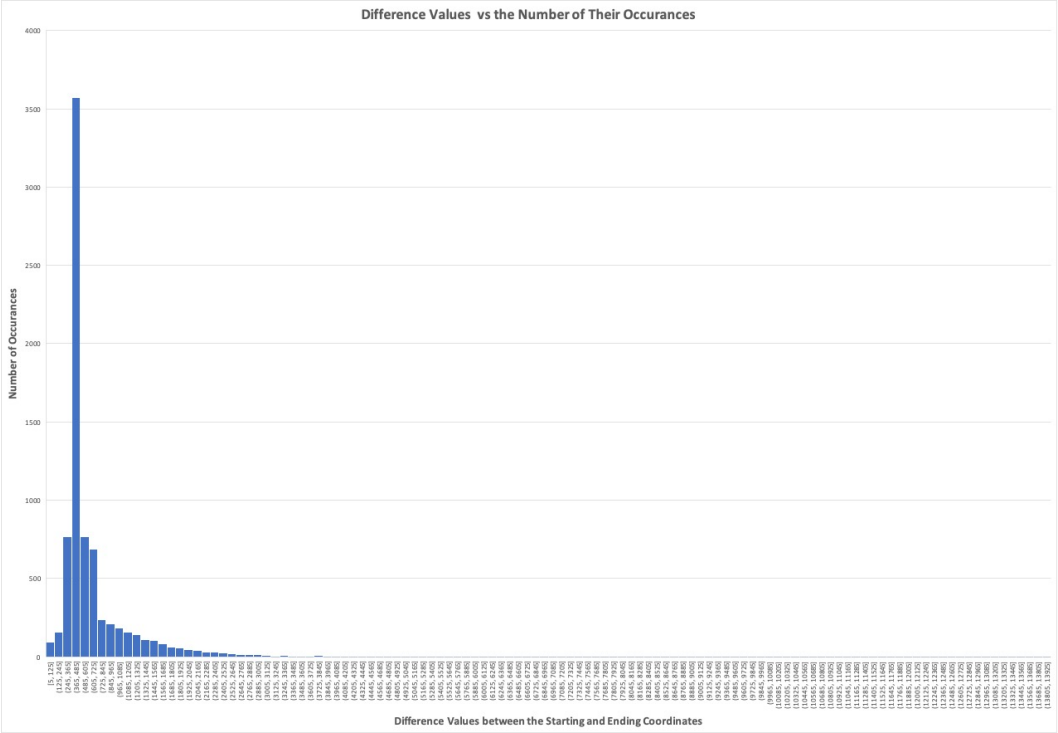

Figure 1: Number of occurrences of different lengths of non-bound regions.

Algorithm 6 Additional code added to *Parser* for non-bound regions

```
1:  function parser(attribute_name, input, output, csvFile)
2:      new_chromStart ← []
3:      new_chromStop ← []
4:      for i < length(chromStart) do
5:          m ← chromStart[i]
6:          diff ← chromStop[i] − m
7:          if diff == 10 then
8:              add chromStart[i] to new_chromStart
9:              add chromStop[i] to new_chromStop
10:         else
11:             if diff < 426 and diff > 390 then
12:                 while diff > 10 do
13:                     add m to new_chromStart
14:                     add m + 10 to new_chromStop
15:                     m ← m + 1
```

16: $diff = chrom\_stop[i] - m$

17: *attribute* (new_chromStart, new_chromStop, output, attribute_name, csvfile)

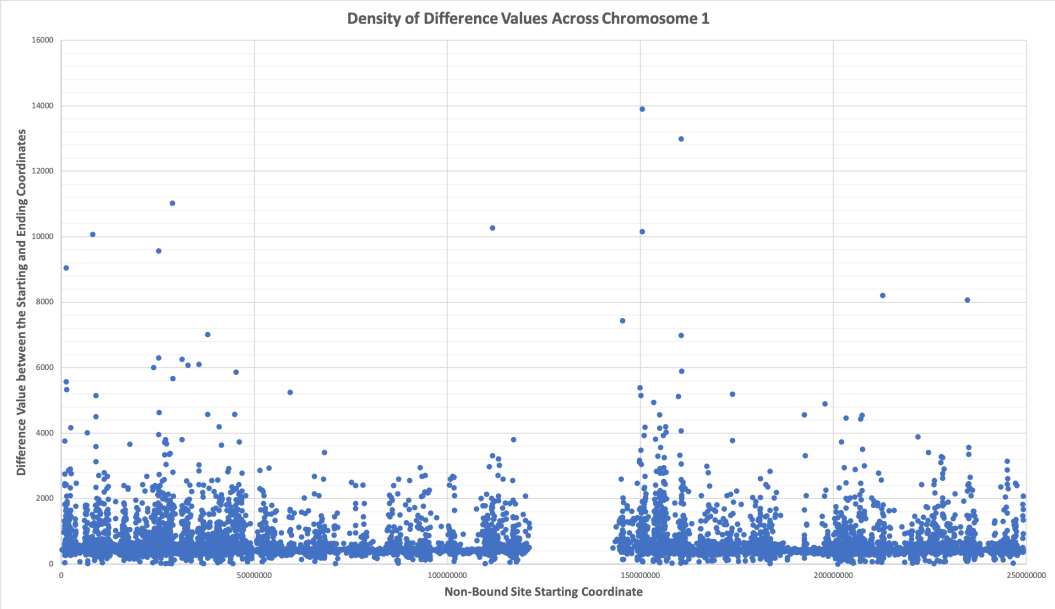

Figure 2: The density of different non-bound region lengths along Chromosome 1.

2.3 Training the Machine Learning Classifier

2.3.1 Choice of Machine Learning Classifier

To distinguish between the bound and unbound regions, the machine learning classifier to be trained was picked to be Random Forest, which are an ensemble of decision trees. Random Forests are powerful classifiers that can solve classification problems, work well with large data sets, and handle multiple features. Moreover, by utilizing multiple decision trees and various sub-samples of the data-set, over-fitting can be avoided. Knowing that Random Forests branch on discriminative features, Random Forests are guaranteed to accurately model a classification problem.

The variables which can be varied for this classifier are inclusive of: number of estimators (*n_estimators*) which determines the number of decision trees in the forest, maximum depth (*max_depth*) which decides the maximum depth of the trees, maximum features (max_features) which is the number of maximum features considered for spitting a node, and random state (random_state), which when set to an int value, ensures outcome consistency across different calls.

2.3.2 Preprocessing Data

Prior to training the machine learning classifier, all the data for bound and non-bound regions were concatenated in one large text file. Moreover, the label which had been set

to true for bound regions was changed to 0 and the label for the non bound regions which had been set to false was set to 1. The formatting of the final text file has been depicted in Table 4.

ChromStart	ChromEnd	Sequence	MGW 1:10	ProT 1:10	HelT 1:10	Roll 1:10	Label
840970	840980	ACAGATAAAG	MGW Val	ProT Val	HelT Val	Roll Val	0
959728	959738	GGAGATAAGG	MGW Val	ProT Val	HelT Val	Roll Val	0
1045283	1045293	TCTTATCAGG	MGW Val	ProT Val	HelT Val	Roll Val	0

Table 4: Final formatting of the text file which its data is processed by the ML classifier

Consequently, the appropriate X matrix and Y vector were created by implementing Algorithm 7, which takes the data text file as an input. To pre-process both the nominal (DNA sequences) and numerical values of the X matrix, the *DataFrameMapper*, *LabelBinarizer*, and *Preprocessing* packages were imported from *sklearn*. More specifically, the *LabelBinarizer* package was used to convert DNA sequences to their binary representations, and the *Preprocessing* package was used to scale the numerical values between -1 to 1. The *DataFrameMapper* class was called to only run the *LabelBinarizer* class on column 2 which held the DNA sequences, and implement *Preprocessing* for columns 3 to 42.

Afterwards, by using *sklearn's Preprocessing* package, the data was split at random into one set of X_train, Y_train, containing 80% of the instances to be used for training and validation, and a testing set, X_test, Y_test containing the other 20% of data. Lastly, the Random Forest classifier with 256 estimators, maximum tree depth of 28, and random state set to 0 was run on training data, keeping all the physical features.

Algorithm 7 Creating the X matrix and Y vector

function *readXY*(filename)
    $X \leftarrow []$
    $Y \leftarrow []$
Require: $filename$ is open
    Read the lines
    for line in lines **do**
        Strip and split the values in each line by white space
        $x \leftarrow$ the stripped and split line
        $Xi \leftarrow x[2:43]$
        $y \leftarrow x[43]$
        append Xi to X
        append y to Y
    return X, Y ▷ X and Y are numpy arrays
Ensure: $filename$ is closed

3 Results

Multiple metrics were used to evaluate the trained model, including classification accuracy, 10-fold cross validation, logarithmic loss and F1 score.

3.1 Classification Accuracy and 10-Fold Cross Validation

After training the data using X_train, Y_train, the output (being either 0 or 1 depending on if the protein sequence belongs to the bound or non-bound region) of X_test was predicted as y_pred and was compared to y_test. The accuracy was calculated to be 0.9999036485097637. To picture this result better, and to ensure that both types of outputs were included in y_pred, which is the vector holding the predicted output of X_test, the confusion matrix was determined which outputs the number of true positives, true negatives, false positives and false negatives in form of a matrix, as shown below. This step was performed since the number of data points for non bound regions were substantially higher than bound regions.

$$\begin{bmatrix} i_{00} & i_{01} \\ i_{10} & i_{11} \end{bmatrix} = \begin{bmatrix} 2309 & 0 \\ 3 & 28824 \end{bmatrix} \tag{1}$$

Here, i_{00} is the predicted number of bound regions which were actually bound regions (true positives), i_{01} is the predicted number of non-bound regions which were actually bound regions (false positives), i_{10} is the number of bound regions which were not predicted as bound regions (false negatives), and i_{11} is the number of predicted non-bound regions which were actually non-bound regions (true negatives). The accuracy from the confusion matrix can be calculated as:

$$Accuracy = \frac{\text{True Positives} + \text{False Negatives}}{\text{Total Number of Samples}} \tag{2}$$

which results in an accuracy of 0.9999036485097637 as mentioned previously.

Moreover, 10-fold cross-validation was performed on the training data, and the average score came out to be 0.9992291710560209.

3.2 Logarithmic Loss

This metric has been designed to penalize false classifications and its value can be anywhere from 0 to infinity. The value of logarithmic loss is calculated as:

$$\text{Logarithmic Loss} = \frac{-1}{N} \sum_{i=1}^{N} \sum_{j=1}^{M} y_{ij} * \log(p_{ij}) \tag{3}$$

where y_{ij} and p_{ij} depict if sample i and the probability of sample i belong to class j respectively. Therefore, the closer the logarithmic loss is to zero, the higher the accuracy. The logarithmic loss for the presented model has been calculated to be 0.00332786257659183, which indicates that the classifier is accurate in distinguishing between bound and non bound regions.

3.3 F1 Score

This metric determines the preciseness and robustness of the trained classifier. The F1 score can be calculated using the following formulas:

$$F1 = 2 * \frac{1}{\frac{1}{\text{precision}} + \frac{1}{\text{recall}}} \tag{4}$$

where

$$Precision = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}} \tag{5}$$

$$Recall = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}} \tag{6}$$

The higher the F1 score, the better the performance of the classier. The F1 score was calculated to be 0.9999479627413228.

```
/Users/yasminsalehi/anaconda3/bin/python3.6 /Users/yasminsalehi/PycharmProjects/Assignment1/ml_classifier.py
y_test: [1 1 1 ... 1 1 1]
Number of 0s:  2312
Number of 1s:  28824
Accuracy:  0.9999036485097637
10-Fold Cross Validation Scores:  [0.99903652 1.         1.         0.99855477 0.99903652 0.99951821
 0.99951821 0.99903642 0.99855463 0.99903642]
Average of the 10-Fold Cross Validation Scores 0.9992291710560209
Log loss:  0.0033278625765918277
Confusion Matrix:  [[ 2309     0]
 [    3 28824]]
f-1 Score:  0.9999479627413228

Process finished with exit code 0
```

Figure 3: The accuracy and precision scores of the trained classifier

Lastly, to determine the important of "sequence readout" in distinguishing the bound from non bound regions, the DNA sequence were excluded from the X matrix and the program was run again. Interestingly, the accuracy of the model did not change by much. Knowing that the Random Forest algorithm branches on discriminative features, it can be interpreted that "shape readout" plays a key role in distinguishing bound from non bound regions.

4 Discussion

This paper presented an approach which could be implemented to distinguish between bound and non-bound regions using DNA sequences' physical properties after training a Random Forest classifier. The accuracy of the predictions came out very high which implied the significance of "shape readout" in determining between bound and non-bound regions.

While carrying out this project, difficulties where encountered while determining the physical properties of the DNA sequences for non-bound regions due to the large run time. This problem was overcome by filtering the data. However, this may have affected the results by biasing the data set. To improve this, faster computers may be used for mapping

DNA sequences to their corresponding physical properties such that data points for the non-bound regions can be selected more randomly from a larger data set. For this experiment, it should be kept in mind that sampling the non-bound regions is very important as the number of bound regions are noticeably less than the non-bound ones. An alternative approach would be adding more data points associated to the bound regions by repetitively feeding the X matrix with the same set of data until the number of data points corresponding to different output classes are balanced. However, this may slow down the training time of the classifier. For future work, one may test the prediction power of this trained classifier for other chromosomes, and train the model with a less biased data-set by exploiting faster computers.

References

[1] Human genome assembly hg19. http://hgdownload.soe.ucsc.edu/goldenPath/hg19/bigZips/chromFa.tar.gz. Accessed: 2018-11-20.

[2] Kirill Batmanov and Junbai Wang. Predicting variation of dna shape preferences in protein-dna interaction in cancer cells with a new biophysical model. *Genes*, 8(9):233, 2017.

[3] Mathieu Blanchette. Positive regulatory regions. http://www.cs.mcgill.ca/~blanchem/561/wgEncodeRegTfbsClusteredV3.GM12878.merged.bed.gz. Accessed: 2018-11-19.

[4] Mathieu Blanchette. Transcription factor binding sites. http://www.cs.mcgill.ca/~blanchem/561/factorbookMotifPos.txt.gz. Accessed: 2018-11-20.

[5] Stephen P Hancock, Tahereh Ghane, Duilio Cascio, Remo Rohs, Rosa Di Felice, and Reid C Johnson. Control of dna minor groove width and fis protein binding by the purine 2-amino group. *Nucleic acids research*, 41(13):6750–6760, 2013.

[6] Rohs Lab. Dna shape. {http://rohsdb.cmb.usc.edu/GBshape/cgi-bin/hgTables}. Accessed: 2018-11-20.

[7] Anthony Mathelier, Beibei Xin, Tsu-Pei Chiu, Lin Yang, Remo Rohs, and Wyeth W Wasserman. Dna shape features improve transcription factor binding site predictions in vivo. *Cell systems*, 3(3):278–286, 2016.

[8] Christophe Oguey, Nicolas Foloppe, and Brigitte Hartmann. Understanding the sequence-dependence of dna groove dimensions: implications for dna interactions. *PloS one*, 5(12):e15931, 2010.

[9] Remo Rohs, Sean M West, Alona Sosinsky, Peng Liu, Richard S Mann, and Barry Honig. The role of dna shape in protein–dna recognition. *Nature*, 461(7268):1248, 2009.