Data294P First Draft Presentation

AN DL APPROACH ON PEAK RECOGNITION IN DNA SEQUENCE

XIAOFEI TENG

Background Introduction

This project aims to identify peaks (high accessibility fragments on a DNA sequence), which are associated with condensate formation within Liquid-Liquid Phase Separation (LLPS), through a direct deep learning approach to learn the pattern of the original DNA sequence (the ATCGs).

The overall idea is to do a two-step approach:

- Step 1: generate a label for each peak as yes/no to being a target peak
- Step 2: feed the DNA sequences of the peaks together with the labels generated in step 1 to a deep neural network and train a model

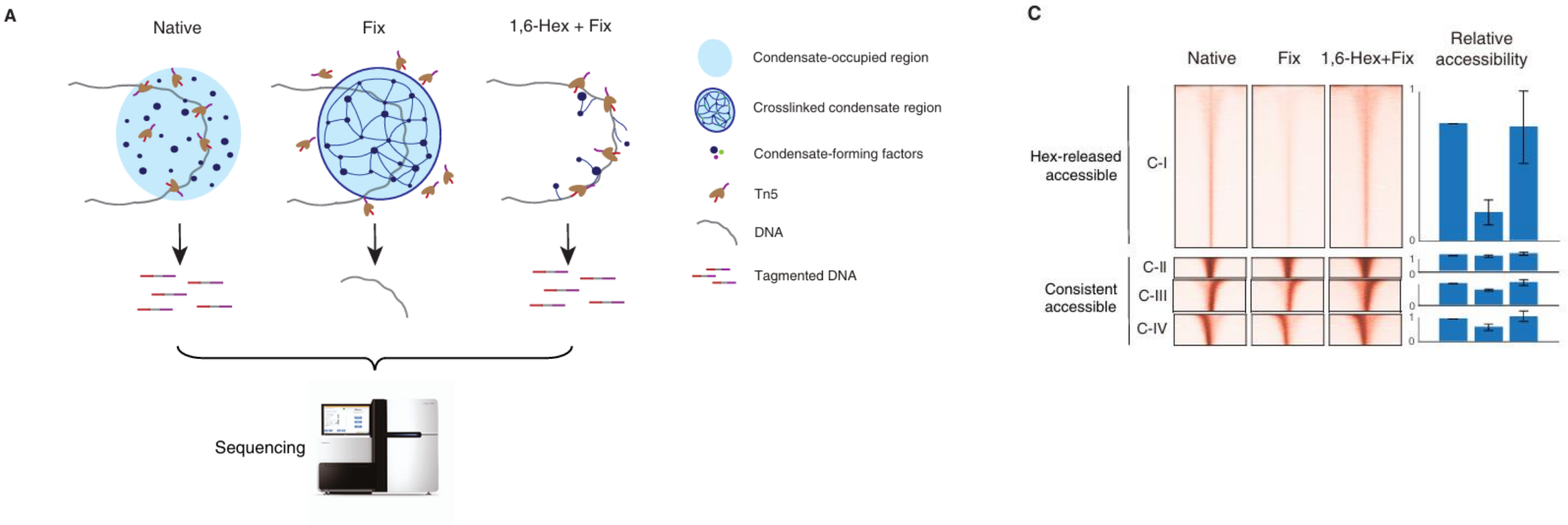
Because the quality of the label will directly influence the outcome of learning, we have to be very careful when selecting methodology for step 1.

Background Introduction

One important feature of the peaks we are looking for is that their responses under a specific set of experimental conditions show **a certain pattern**.

For the three experimental conditions, namely native condition (control group), chemical fixation, and chemical fixation with 1,6-hexanediol, **the accessibility read is significantly lower for the fixation group comparing to the other two groups**, which no significant difference is observed. This is because under chemical fixation, the transcriptional components form multivalent, dynamic, intermolecular inter actions that are sensitive to disruption by 1,6-hexanediol[(1)].

In conclusion, we are looking for a **V-shape** pattern across the 3 experimental conditions.


A
Native
Fix
1,6-Hex + Fix
Condensate-occupied region
Crosslinked condensate region
Condensate-forming factors
Tn5
DNA
Tagmented DNA
Sequencing
C
Native
Fix
1,6-Hex+Fix
Relative accessibility
Hex-released accessible
C-I
Consistent accessible
C-II
C-III
C-IV
1
0

ATAC-seq Technique

The ATAC-seq technique enables the genome-wide detection of chromatin accessibility and plays a crucial role in various fields such as gene regulation and disease mechanisms. For an individual experimental treatment, raw data produced by ATAC-seq can be converted into a vector using some bioinformatics software. The dimension of the vector is the number of **open regions or peaks**, with each component representing a peak and its value indicating the **relative quantification** of accessibility of that peak.

Because only relative quantifications are recorded, only comparison within the same experimental group is meaningful. Thus, to compare between different groups, normalization is necessary.

Data Summary

The dataset is consist of 2 subsets, one of which will be used to make labels, the other to train the model.

The first subset only has 1 table, which has 68809 rows, each row represents a peak of length 400 and has 6 columns:

- chrom: the name of the chrome the peak is on
- start: the start point of the peak
- end: the end point of the peak
- Fresh: the accessibility read for the control group
- Fix: the accessibility read under chemical fixation
- HexFix: the accessibility read under chemical fixation with 1,6-hexanediol

The second subset has 2 tables, each has 137923 rows; each row represents a peak of length 600. The first table in this subset has the following columns:

- chrom: the name of the chrome the peak is on
- start: the start point of the peak
- end: the end point of the peak
- id: the name of peak
- signal: a signal strength calculated by a bio-info specific software when generating the sequence
- label: labels generated from the first step
- intensity: the average intensity read for the sequence (the true intensity read of a sequence should be a distribution with one read for each nucleotide, but here I just get the average over the distribution)

The second table has the following columns:

- id: the name of the peak
- sequence: the actual sequence (consist of A, T, C, G, and N) of the peak, length is 600

Note that the peaks of the two subsets are not of a 1-to-1 relationship. The peaks in subset 1 underwent a special treatment called overlapping (which is not done by me), and as a result, peaks in subset 1 could correspond to one or several peaks in subset 2, while some peaks in subset 2 are not included in subset 1 at all.

Project Outline

- Data normalization
- Data transformation
- Clustering algorithm (Kmeans) to generate label
- Data cleaning
- One-hot encoding DNA sequence
- Train a CNN model using the one-hot encoded sequence and generated label

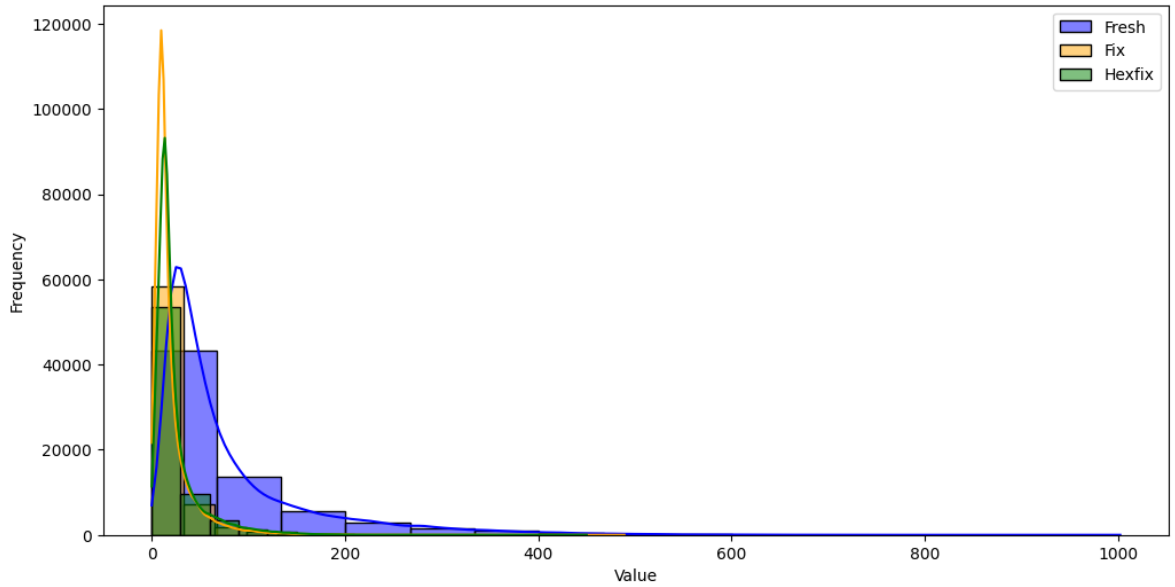
Data Normalization

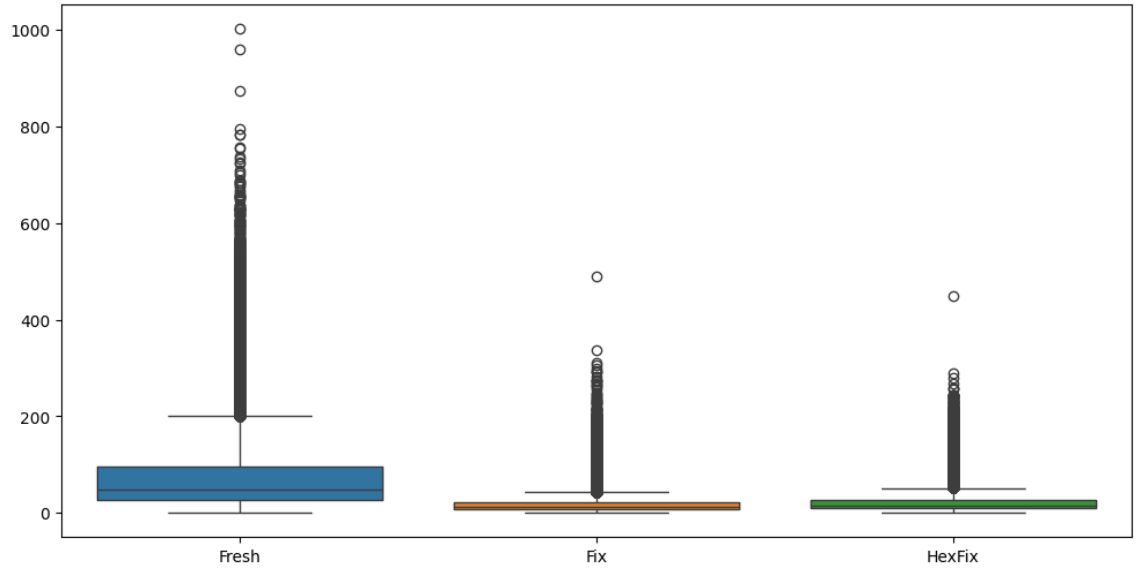
As mentioned before, proper normalization is necessary before comparison between groups. But even before that, let us take a closer look at our data:

	chrom	start	end	Fresh	Fix	HexFix
0	chr10	3117593	3117993	44	10	10
1	chr10	3172364	3172764	23	13	11
2	chr10	3172903	3173303	72	28	18
3	chr10	3181097	3181497	15	12	8
4	chr10	3191617	3192017	20	10	4
...	...	...	...	...	...	...
68804	chrY	90807654	90808054	94	58	40
68805	chrY	90808489	90808889	92	23	34
68806	chrY	90810605	90811005	83	33	28
68807	chrY	90825230	90825630	42	6	17
68808	chrY	90828665	90829065	55	11	24

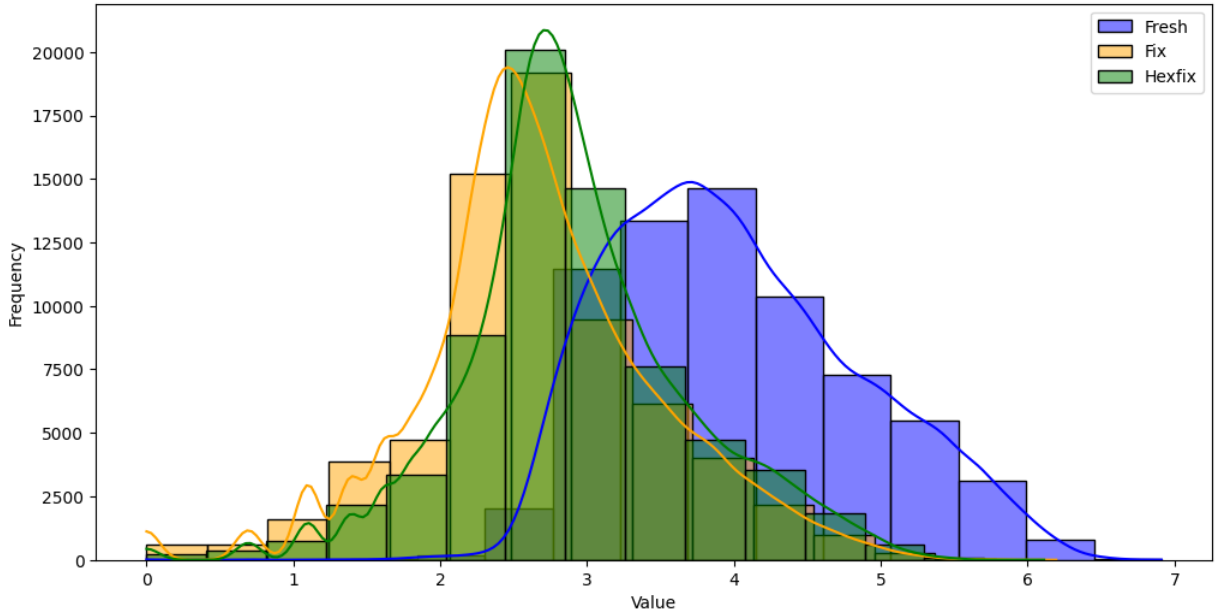
68809 rows × 6 columns

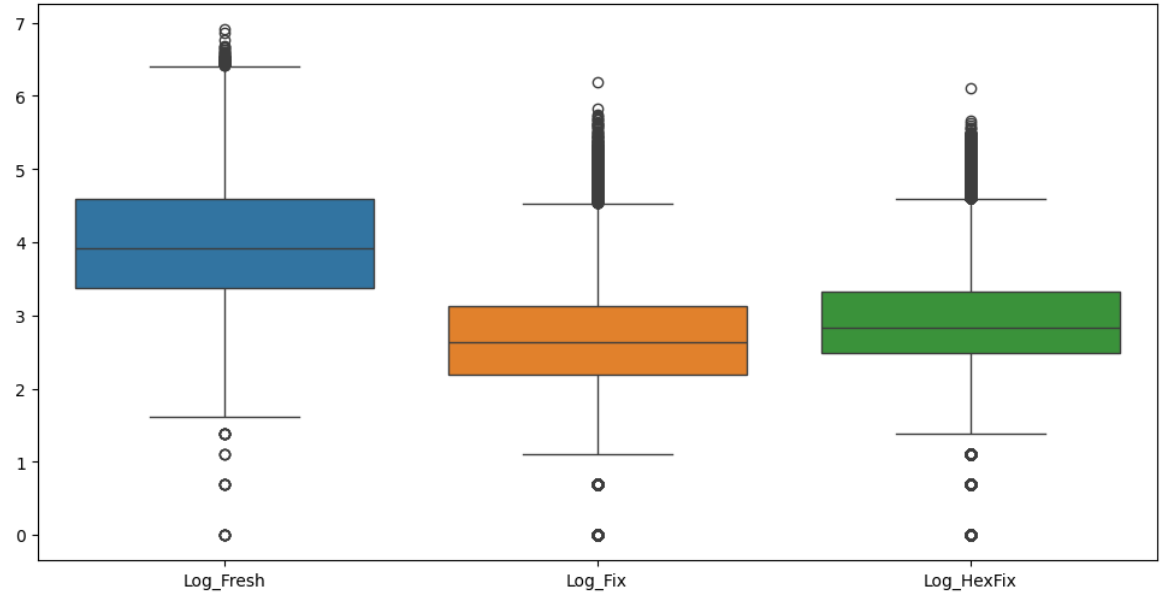
From both the histogram and the box plot we can see that the data is very skewed to the right, and also the three columns are of different “scale”.



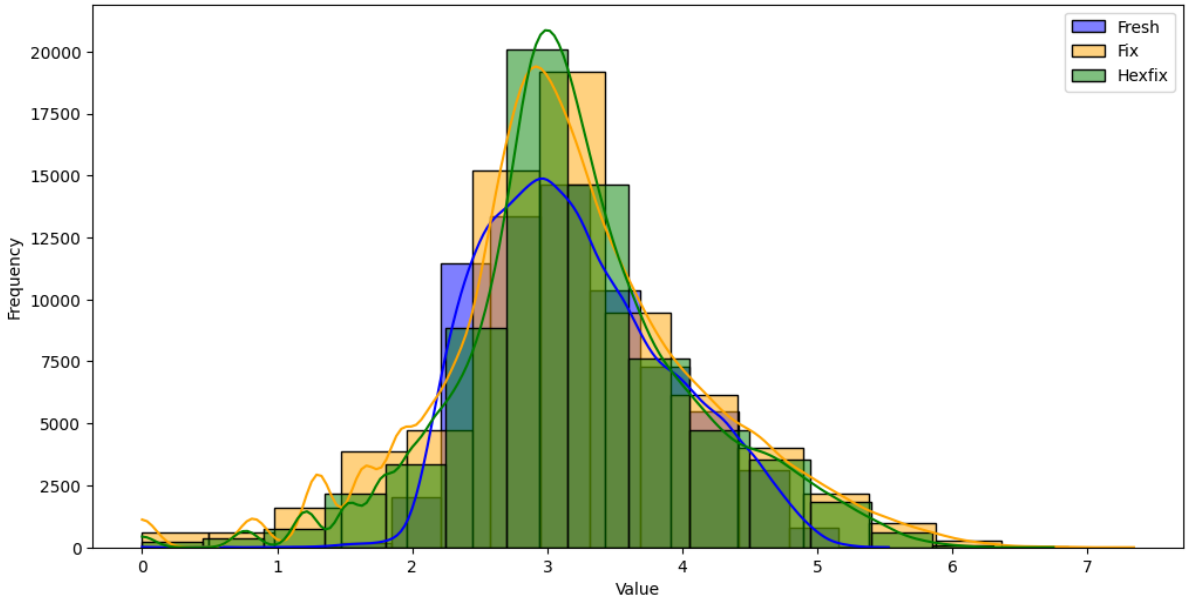


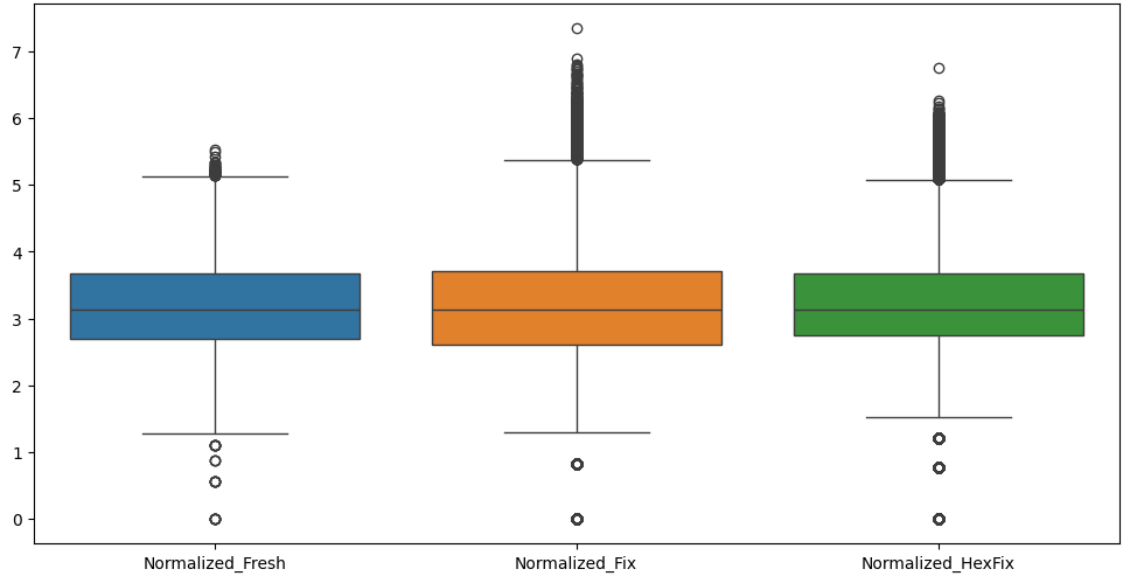
One common approach to deal with data with a long tail is to apply a log transformation. Here, a log2 transformation is used. Can see the data become more “normal” after applying the log transformation.





In the field of biological information, quantile normalization is commonly used[2]. In this project, Median normalization (MED), or the 50-quantile normalization is used. This method lines up the median of each column to make the data be on the same “scale”.





Data Transformation

Now we have 3 normalized columns. Before we feed the data to a clustering algorithm to generate the labels, it is better to apply certain transformations that better capture the feature we care about, and hopefully could improve the clustering result.

Some possible transformation methods are:

- Z-score transformation (does not do a significantly better job to capture wanted features, and might have problems)
- Difference transformation (simple and effective)
- HSV transformation

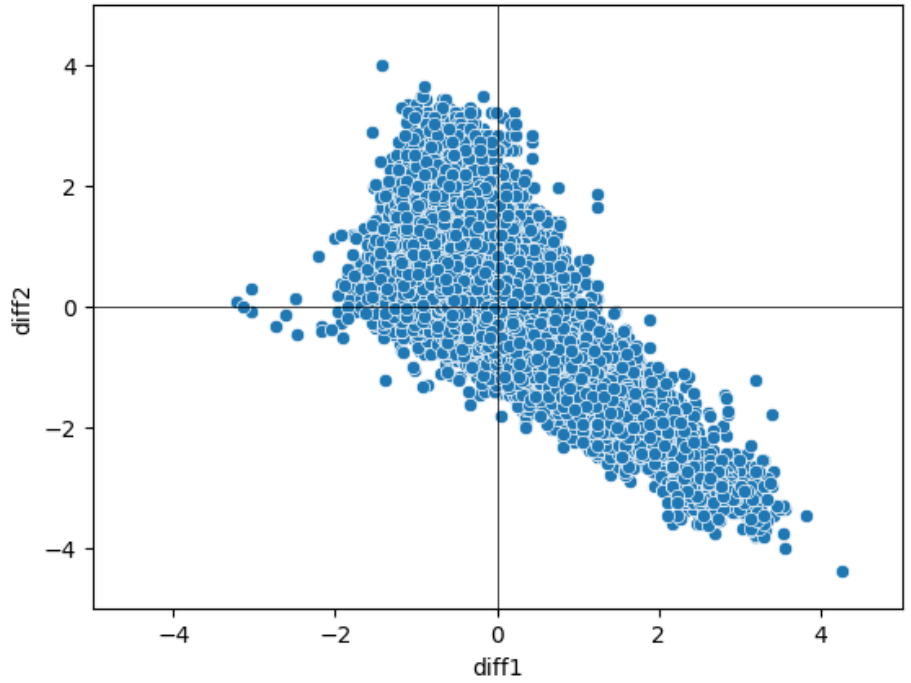
Difference transformation

Transform (Fresh, Fix, HexFix) to (Fresh - Fix, Fix - HexFix)

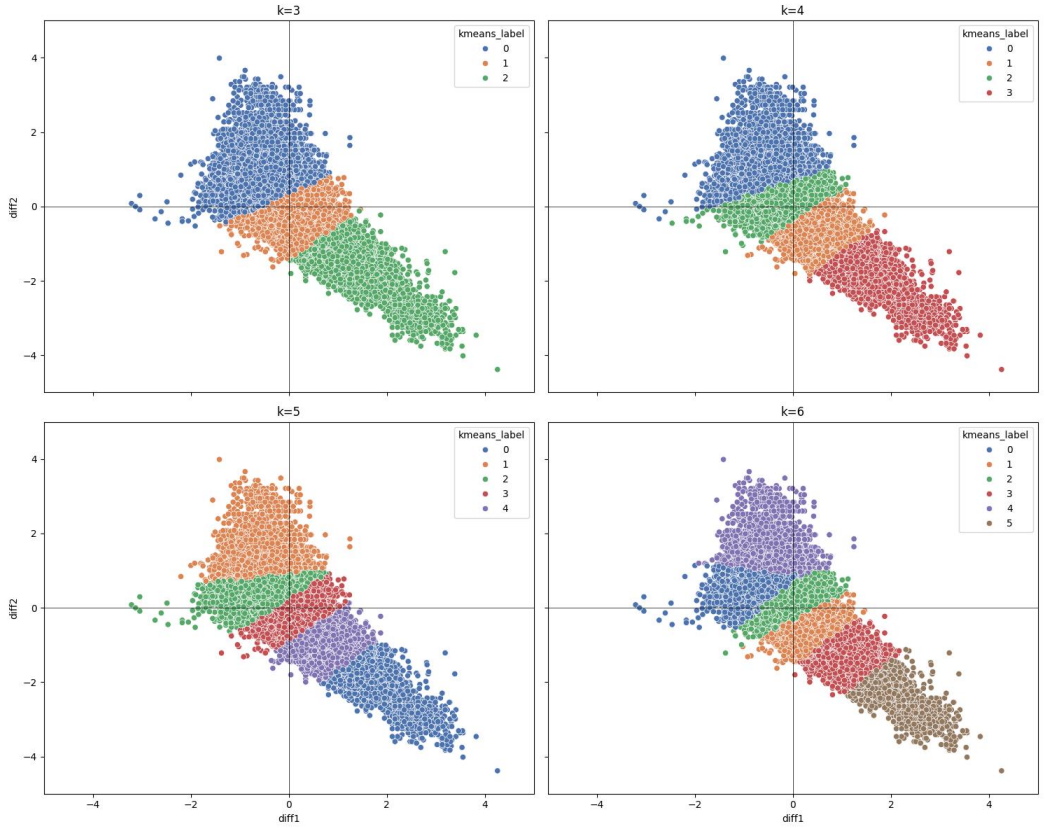
- Advantage: focuses on the difference feature we care about, dimension reduction, visualization possible
- Disadvantage: loss of information

Fresh	Fix	HexFix
3.043850	2.842246	2.647471
2.541208	3.128098	2.743539
3.430700	3.991285	3.250900
2.216993	3.040257	2.425915
2.434435	2.842246	1.776950
...	...	...
3.641332	4.833141	4.100086
3.624318	3.766975	3.925393
3.542931	4.179826	3.717768
3.007498	2.306504	3.191205
3.218717	2.945382	3.553901

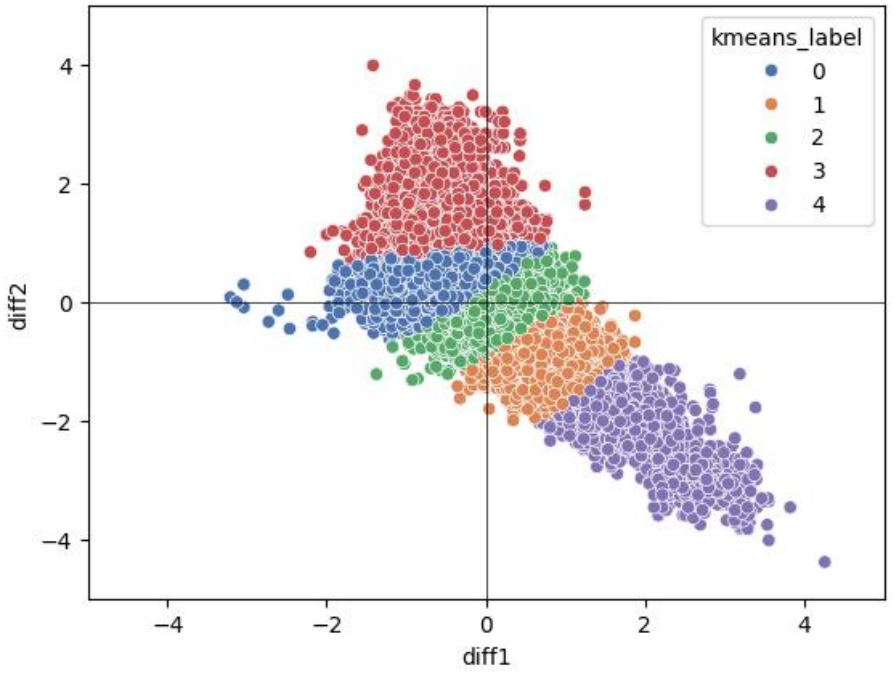
diff1	diff2
0.201604	0.194775
-0.586890	0.384559
-0.560585	0.740385
-0.823264	0.614342
-0.407812	1.065296
...	...
-1.191810	0.733056
-0.142657	-0.158418
-0.636895	0.462058
0.700994	-0.884701
0.273335	-0.608519



Kmeans



Kmeans result for k=5



Mean of each label

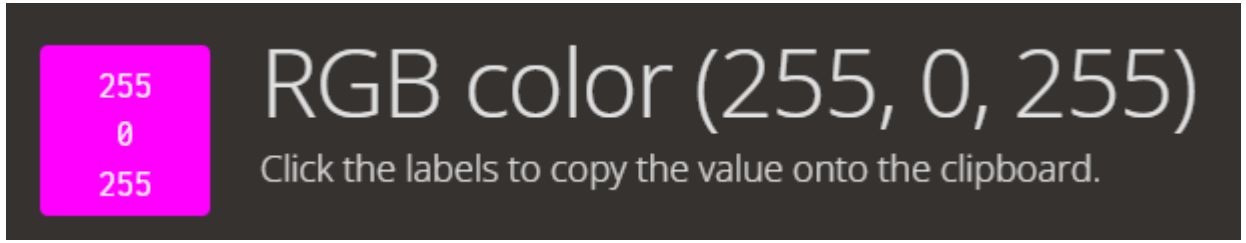
kmeans_label	diff1	diff2
0	-0.432720	0.406446
1	0.713944	-0.942020
2	0.087102	-0.208641
3	-0.549190	1.279988
4	1.739027	-2.016596

```
len(df[df['kmeans_label'] == 4])
```

```
3492
```

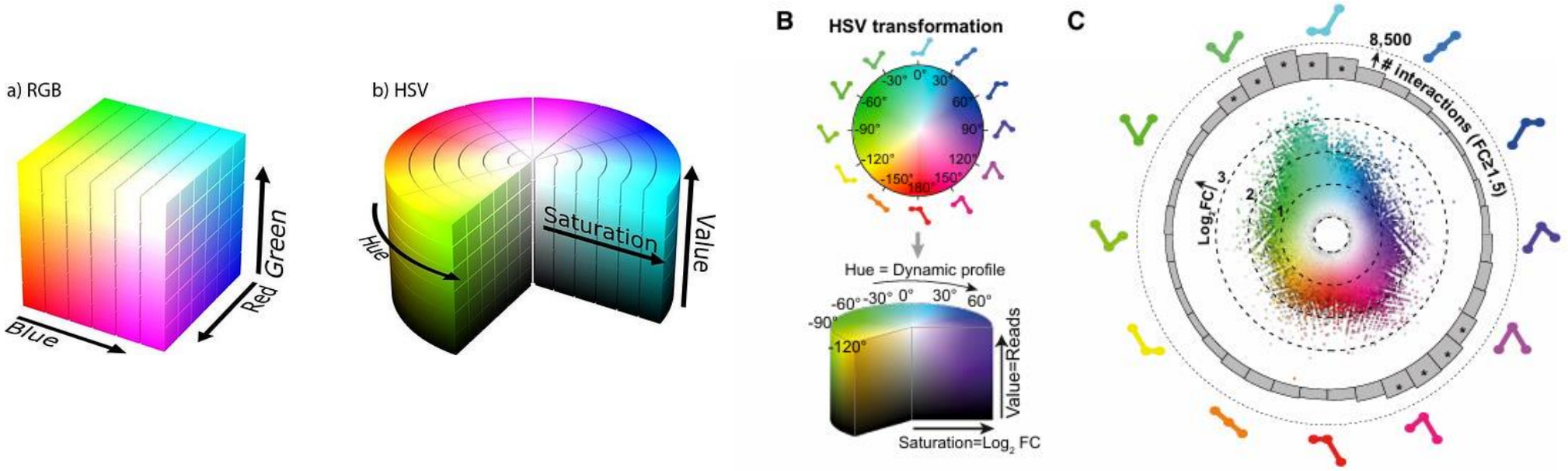
HSV transformation

Say we have a tuple (R, G, B) representing a color. Imagine you have a V-shape pattern for the RGB values (high R and B value, low G value), what color could it possibly look like?



Could it be that the V-shape pattern we are looking for is simply within a specific color range when converted into a RGB color?

Indeed this is the case! If we treat the (Fresh, Fix, HexFix) tuple as RGB values of a color, all V-shape tuples should represent a color within a certain color range, in other words, a color which hue is in a certain range (when transform a RGB color into a HSV color).



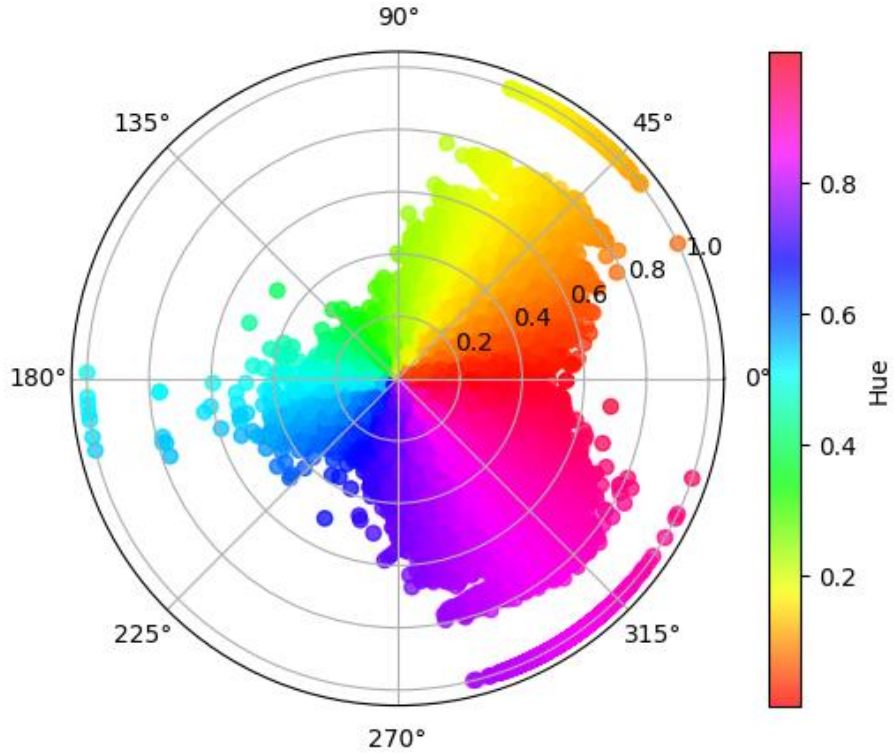
Siersbæk et al., 2017. [3]

Fresh	Fix	HexFix
0.550832	0.387107	0.392360
0.459871	0.426039	0.406597
0.620838	0.543603	0.481789
0.401199	0.414075	0.359525
0.440549	0.387107	0.263347
...	...	...
0.658955	0.658261	0.607640
0.655877	0.513052	0.581750
0.641148	0.569282	0.550980
0.544253	0.314140	0.472942
0.582477	0.401153	0.526694

The columns are normalized to be between [0, 1] for convenience of HSV transformation

H	S	V
0.994652	0.297233	0.550832
0.060823	0.115845	0.459871
0.074091	0.223970	0.620838
0.206006	0.131740	0.414075
0.116402	0.402229	0.440549
...	...	...
0.164412	0.077874	0.658955
0.919834	0.217761	0.655877
0.033829	0.140636	0.641148
0.884983	0.422806	0.544253
0.884607	0.311297	0.582477

HSV values are also between [0, 1]

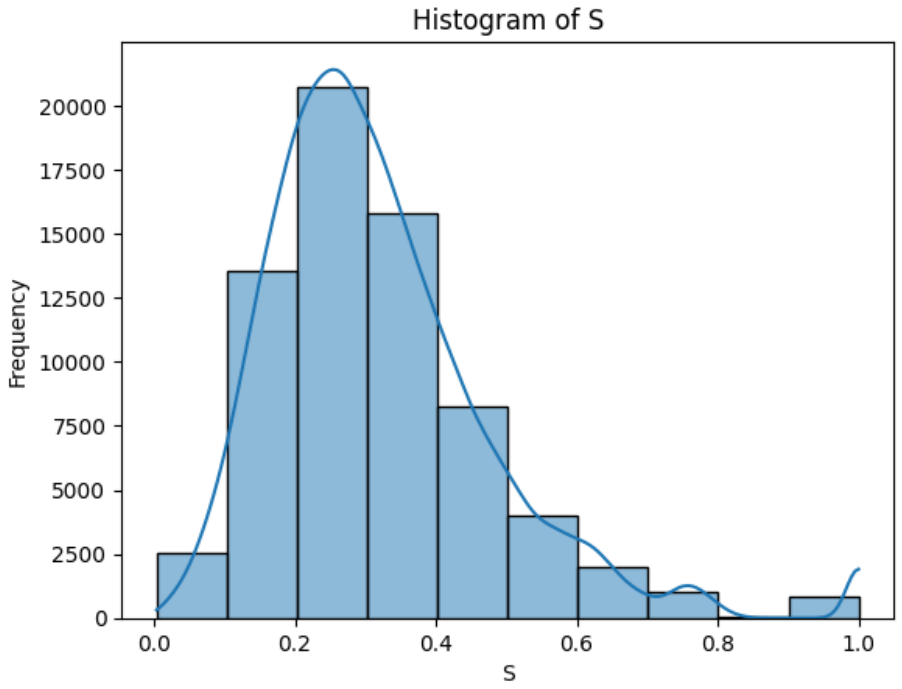


Interpretation of HSV transformation

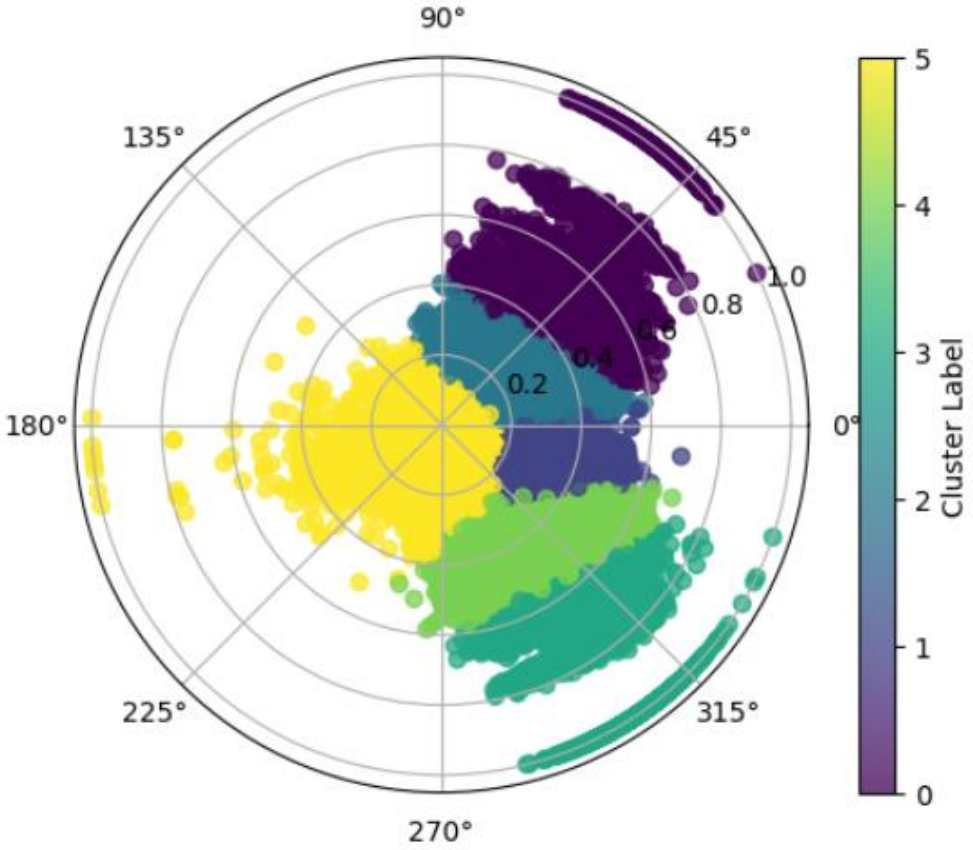
For a (R, G, B) to (H, S, V) transformation:

- H is the “shape” of the R-G-B angle
- S is (max – min) / max, where max, min are the maximum, minimum value in (R, G, B)
- V is the max value in (R, G, B)

Except for certain H range (the V-shape), we also look for certain S range (how deep the V is).



Kmeans



Mean of each label

hsv_label	Fresh	Fix	HexFix
0	0.496530	0.421150	0.232232
1	0.637153	0.454633	0.519214
2	0.579010	0.486980	0.427054
3	0.497304	0.141714	0.456892
4	0.551271	0.321013	0.495641
5	0.580395	0.513293	0.552799

Count of each label

```
hsv_label
1    18744
2    17179
5    12801
4    12256
0     4508
3     3321
```

Data cleaning

The second subset comes in a very strange format (at least strange to me): .bed and .fa file.

- .bed file can be read by pandas using the read_csv function and use '\t' as delimiter.
- .fa file is generated by a software called FASTA, which is wildly used in bioinformatics to store nucleotide or protein sequences. In python, it has to be read using special package functions (SeqIO function from Bio package).
- This is what the dataset looks like, it is thus easy to see that the two tables can join on a common key, id:

	chrom	start	end	id	signal	label	intensity
0	chr10	3117369	3117969	mESC_Fresh_rep_atac_peak_9006	13.17310	3	7.712389
1	chr10	3164504	3165104	mESC_Fresh_rep_atac_peak_9007	7.78068	6	6.642889
2	chr10	3172174	3172774	mESC_Fresh_rep_atac_peak_9008	7.78068	1	7.187947
3	chr10	3172812	3173412	mESC_Fresh_rep_atac_peak_9009	38.11530	1	11.254194
4	chr10	3180991	3181591	mESC_Fresh_rep_atac_peak_9010	2.42006	1	7.906741
...	...	...	...	...	...	...	...
137918	chrX	167063692	167064292	mESC_Fresh_rep_atac_peak_137919	8.80371	3	13.705830
137919	chrX	167067821	167068421	mESC_Fresh_rep_atac_peak_137920	12.04240	1	9.497028
137920	chrX	167091722	167092322	mESC_Fresh_rep_atac_peak_137921	4.91356	2	8.347293
137921	chrX	167157718	167158318	mESC_Fresh_rep_atac_peak_137922	39.56930	6	9.215316
137922	chrX	167158298	167158898	mESC_Fresh_rep_atac_peak_137923	2.42006	3	12.684612

137923 rows × 7 columns

	id	sequence
0	mESC_Fresh_rep_atac_peak_9006	GGTCTCCATGTCTCCATGTGTCCTGGAACACTAAAGGACTACTCTC...
1	mESC_Fresh_rep_atac_peak_9007	GTCTGCGTGGAACGGGTCTCTGGTCACAGGTGGGCAAGGAGTTTGA...
2	mESC_Fresh_rep_atac_peak_9008	GTAGAAAGAACATGAAGCCTGTGTCAGCAACATGATGATGCCTGTG...
3	mESC_Fresh_rep_atac_peak_9009	TGTATGCTGTCTTTCAGTATTGTTTTATTATAAGTGCCTTTAAAGA...
4	mESC_Fresh_rep_atac_peak_9010	ACTTTTGAAGACACTCTGGAAGGATGGGAAACACCCTAATCTGGGT...
...	...	...
137918	mESC_Fresh_rep_atac_peak_137919	CTTTTCTAAAGACACAGGTGTTCCTTAGACCTTTGAAACGGACATA...
137919	mESC_Fresh_rep_atac_peak_137920	AAAGTCTGAGACCAGCCTCTCTATACAGTGAGTTCCAGGGTAACCA...
137920	mESC_Fresh_rep_atac_peak_137921	CACTGTGTAGCACAGGCTAGCCTTAAACTCACTATACAGCTTAGTC...
137921	mESC_Fresh_rep_atac_peak_137922	GGTGGATTTTAAATTATCTTTTACTCGGTGACTCATAGAGACCCAT...
137922	mESC_Fresh_rep_atac_peak_137923	ACATTCAAGATCTATAGCACAGATTTTTTGAGATTATAATATAGTC...

137923 rows × 2 columns

There are some problems to be solved before this data could be used to train a model. First, the labels of the peaks are from 1 to 6, while labels 1 to 5 represent class 0 to 4 in the Kmeans result in former step, while label 6 represents “unlabeled”. Thus, we have to delete all rows with label 6 (this is nearly half of the data), and reduce all label by 1 to get the original label we generated from the clustering algorithm.

I also swap label 0 with label 4, so that label 0 now represents the V-shape we are looking for. This saves time for me to remember which label is the label we target at.

```
label
6    69536
1    21962
3    20938
2    12840
4     9150
5     3497
Name: count, dtype: int64
```

Another problem is that, besides A, T, C, and G, there is also a N nucleotide in the sequence column, which represents “unknown” or “any” nucleotide. We do not want this noise in our model, so I want to delete all rows whose sequence contains the N nucleotide.

This is what the data looks like after data cleaning:

	chrom	start	end	id	signal	label	intensity	sequ
0	chr10	3117369	3117969	mESC_Fresh_rep_atac_peak_9006	13.17310	2	7.712389	GGTCTCCATGTCTCCATGTGTCCTGGAACACTAAAGGACTACT(
1	chr10	3172174	3172774	mESC_Fresh_rep_atac_peak_9008	7.78068	0	7.187947	GTAGAAAGAACATGAAGCCTGTGTCAGCAACATGATGATGCCT(
2	chr10	3172812	3173412	mESC_Fresh_rep_atac_peak_9009	38.11530	0	11.254194	TGTATGCTGTCTTTCAGTATTGTTTTATTATAAGTGCCTTTAA/
3	chr10	3180991	3181591	mESC_Fresh_rep_atac_peak_9010	2.42006	0	7.906741	ACTTTTGAAGACACTCTGGAAGGATGGGAAACACCCTAATCTG(
4	chr10	3191493	3192093	mESC_Fresh_rep_atac_peak_9011	5.83284	3	6.828549	TCTGCTCTAGCAGACTTTCTCATGAATAATGCACTACCCCCCT
...	...	...	...	...	...	...	...	
68370	chrX	167049561	167050161	mESC_Fresh_rep_atac_peak_137918	6.78967	0	16.478917	TGTACCGGGACACACTTTCCCCATCTGTGGCTAAACTCAGTCAC
68371	chrX	167063692	167064292	mESC_Fresh_rep_atac_peak_137919	8.80371	2	13.705830	CTTTTCTAAAGACACAGGTGTTCCTTAGACCTTTGAAACGGAC
68372	chrX	167067821	167068421	mESC_Fresh_rep_atac_peak_137920	12.04240	0	9.497028	AAAGTCTGAGACCAGCCTCTCTATACAGTGAGTTCCAGGGTAA(
68373	chrX	167091722	167092322	mESC_Fresh_rep_atac_peak_137921	4.91356	1	8.347293	CACTGTGTAGCACAGGCTAGCCTTAAACTCACTATACAGCTTA(
68374	chrX	167158298	167158898	mESC_Fresh_rep_atac_peak_137923	2.42006	2	12.684612	ACATTCAAGATCTATAGCACAGATTTTTTGAGATTATAATATA(

68375 rows × 8 columns

While we got rid of nearly half the data, we still have about the same amount of rows in remain comparing to the first dataset, so it’s not too bad.

One-hot encoding

A common approach to extract patterns from a sequence in bioinformatics is to one-hot encode the sequence, so that a sequence of length n becomes a matrix of 4*n. This matrix is then fed into a CNN to extract the spatial information for further analysis. The methodology used in this project has referred to a paper by Han Yuan and David R. Kelley (2022)[4].

This is how the one-hot encoding looks like:

```
df['one_hot']
```

```
0        [[0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,...
1        [[0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1,...
2        [[0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,...
3        [[1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0,...
4        [[0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0,...
                                ...
68370    [[0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0,...
68371    [[0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0,...
68372    [[1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0,...
68373    [[0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0,...
68374    [[1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0,...
Name: one_hot, Length: 68375, dtype: object
```

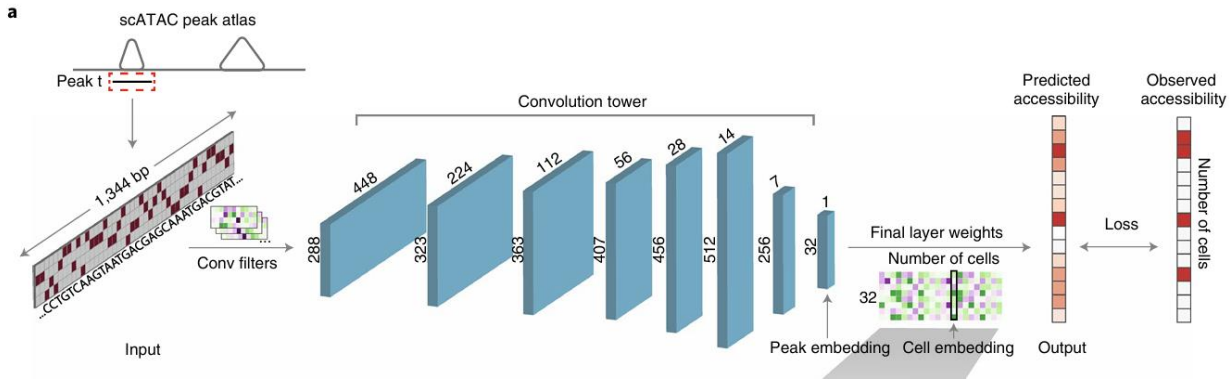
CNN architecture

Because of the property of one-hot encoding, namely that each column can only have on “1” entry, the row-wise relationship (or between-row) is naturally of less significance than column-wise relationship (or within-row). This has led to a certain kind of CNN architecture we use in this project.

Instead of the Conv2d layer, which has a 2d kernel and is usually used on spatial data, here we use Conv1d layer, which has a 1d kernel and is usually used on sequential data. In this specific example, we treat each row as a individual sequence and apply Conv1d layer to it to get a new sequence for each channel. This approach has corresponding bioinformatics significance, for example, the first row could be imagined as the distribution for ‘A’ nucleotide.

CNN architecture

The CNN architecture design also have referred to the paper by Han Yuan and David R. Kelley (2022)[4].




```python
class ConvBlock(nn.Module):
    def __init__(self, in_channels, out_channels, kernel_size, pool_size):
        super(ConvBlock, self).__init__()
        self.conv = nn.Conv1d(in_channels, out_channels, kernel_size, padding=kernel_size // 2)
        self.pool = nn.MaxPool1d(pool_size)

    def forward(self, x):
        x = F.relu(self.conv(x))
        x = self.pool(x)
        return x

class Conv1dCNN(nn.Module):
    def __init__(self, num_classes):
        super(Conv1dCNN, self).__init__()
        self.conv_block1 = ConvBlock(4, 8, 4, 2)
        self.conv_block2 = ConvBlock(8, 16, 4, 2)
        self.conv_block3 = ConvBlock(16, 32, 4, 2)
        self.fc1 = nn.Linear(32 * 75 + 1, 128)
        self.fc2 = nn.Linear(128, num_classes)
        self.dropout = nn.Dropout(0.5)

    def forward(self, x, intensity):
        x = self.conv_block1(x)
        x = self.conv_block2(x)
        x = self.conv_block3(x)
        x = x.view(x.size(0), -1)
        x = torch.cat((x, intensity.unsqueeze(1)), dim=1)
        x = F.relu(self.fc1(x))
        x = self.dropout(x)
        x = self.fc2(x)
        return x
```

Training and evaluation

- Data is randomly sampled from the original dataset with a 80%/20% train/test split ratio
- Default sample size 20000 (16000 training data and 4000 validation data)
- Use cross entropy loss
- Learning rate set to be 0.001, with 1e-4 weight decay
- Train 20 epochs, record the training and validation loss for each epoch, plot the loss when finished
- Evaluate on both the overall accuracy and the accuracy on our target label (label 0 in this case) to prevent the model **from biased of unbalanced label number**

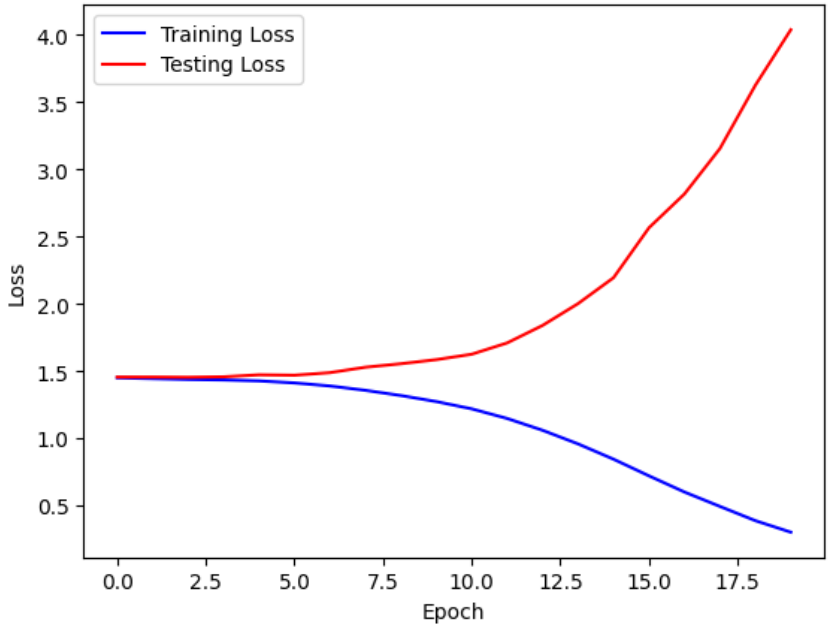
	Positive	Negative
Positive	TP	FP
Negative	FN	TN

```
df['label'].value_counts()
```

```
label
4    21960
2    20933
1    12838
3     9148
0     3496
Name: count, dtype: int64
```

Result

Long story short, the CNN model does not work out well. The validation loss almost always goes up immediately after the training loss starts to go down (as shown below). This is a strong sign of overfitting in ML, where the model simply “remember” the data instead of “learn” the underlying pattern, which makes the model impossible to generalize to unseen data.




```
Overall Accuracy: 0.2542
Correct Label: 1017.0000
Total Samples: 4000.0000
Accuracy on label 0: 0.0830
Correct 0 Label: 19.0000
Total 0 Sample: 229.0000
```

Discussion

Several possible reasons might lead to this result:

- There's a problem with the data
  - The data has too little pattern/too much noise
  - The labels have poor quality
- There's a problem with the model
  - The model is too complicated for the data
  - The model architecture does not fit the data

Future plan

Currently, I am working on testing different hyperparameters as well as different architecture design. By doing so, I hope to rule out the possibility that there is a problem with model selection or the training process.

Meanwhile, my colleagues are working on feature engineering of the original input data, namely on doing experiments to record the complete distribution of the sequences' intensity (we only have the average by now). They believe this should provide some extra information for the model to learn an underlying pattern.

If we somehow manage all these above and it still does not work, we might go back to the part of how we generate the labels. Some new ideas already come up, for example, use a regression model to regress the possibility of a peak being V-shape instead of using clustering algorithms.

Links

- Project repository: https://github.com/N1colTeng/DATA-294P
- Link to presentation video: https://youtu.be/VmiFKyky7bI

Reference

(1) He et al., 2024, Cell. Dual-role transcription factors stabilize intermediate expression levels.

(2) Beibei Wang, Fengzhu Sun & Yihui Luan, 2024. Comparison of the effectiveness of different normalization methods for metagenomic cross-study phenotype prediction under heterogeneity.

(3) Siersbæk et al., 2017, Molecular Cell 66. DynamicRewiring of Promoter-Anchored Chromatin Loops during Adipocyte Differentiation.

(4) Han Yuan and David R. Kelley, 2022. scBasset: sequence-based modeling of single-cell ATAC-seq using convolutional neural networks.