Real Data: 16s rRNA

Yichen Han

2024-08-22

After modifying IG implementation and testing it on synthetic data, we now want to work with real data. We choose a rather simple task as already done by the package dvp-team: 16S rRNA gene detection vs. bacterial genomes.

Pretrained Model

We load the best model as instructed by the online tutorial.

```
## Using checkpoint checkpoints/16S_vs_bacteria_checkpoints/Ep.005-val_loss0.01-val_acc0.999.hdf5
```

```
## [[1]]
## [[1]]$confusion_matrix
##           Truth
## Prediction  16s bacteria
##   16s      1250        1
##   bacteria    0     1249
##
## [[1]]$accuracy
## [1] 0.9996
##
## [[1]]$categorical_crossentropy_loss
## [1] 0.004219269
##
## [[1]]$AUC
## [1] 1
##
## [[1]]$AUPRC
## NULL
```

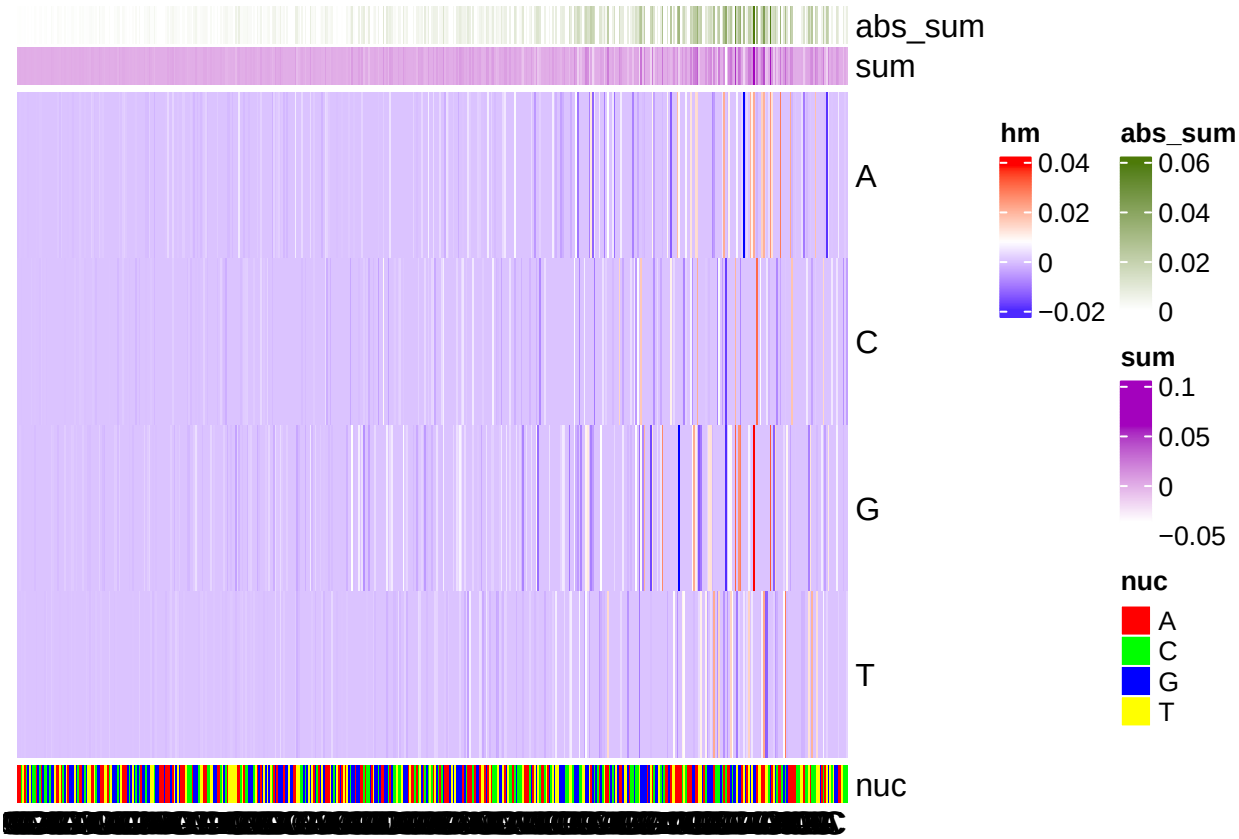
We see that the model has very high accuracy.

We then load a random 16S gene sequence from the validation data, and load a bacteria genome from the validation data as our baseline. We extract positions 499-998 (500 bp) for both sequences.

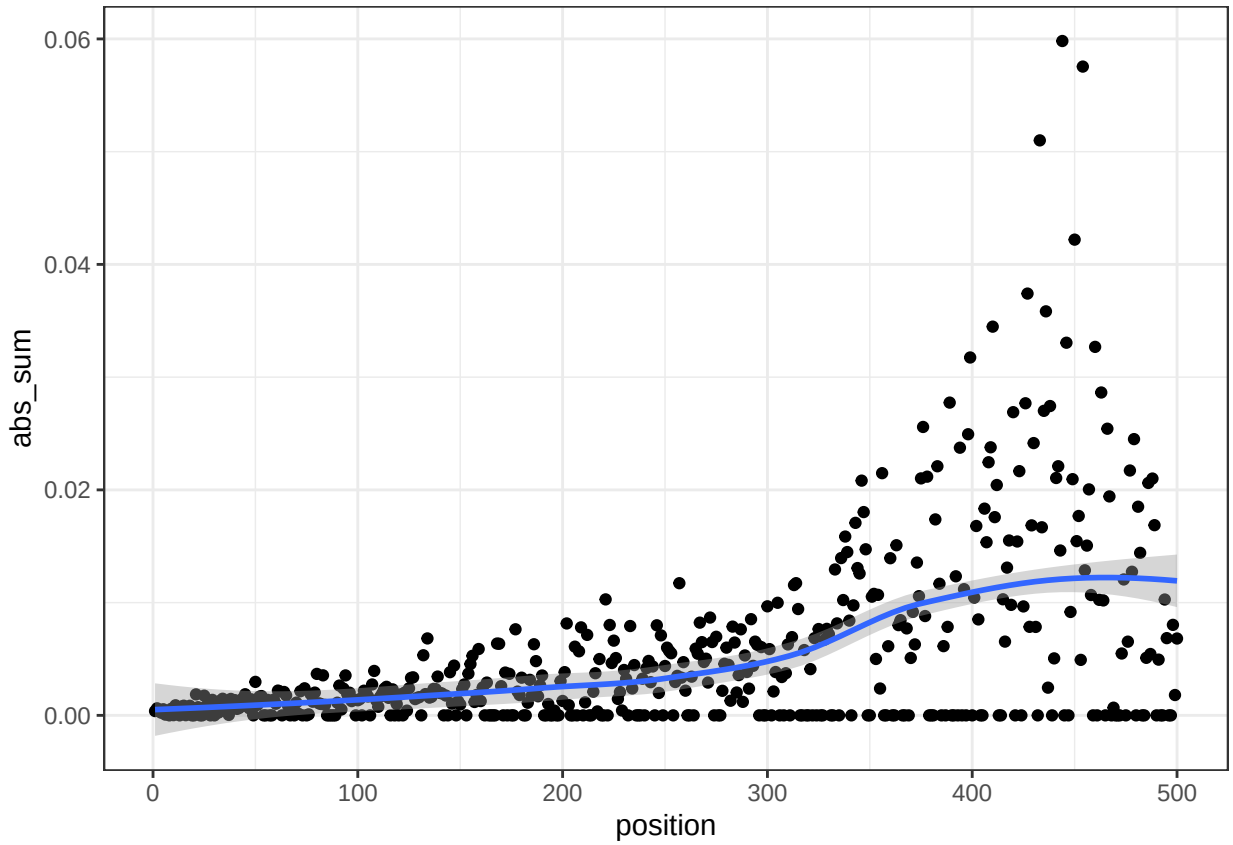
We then perform modified IG.

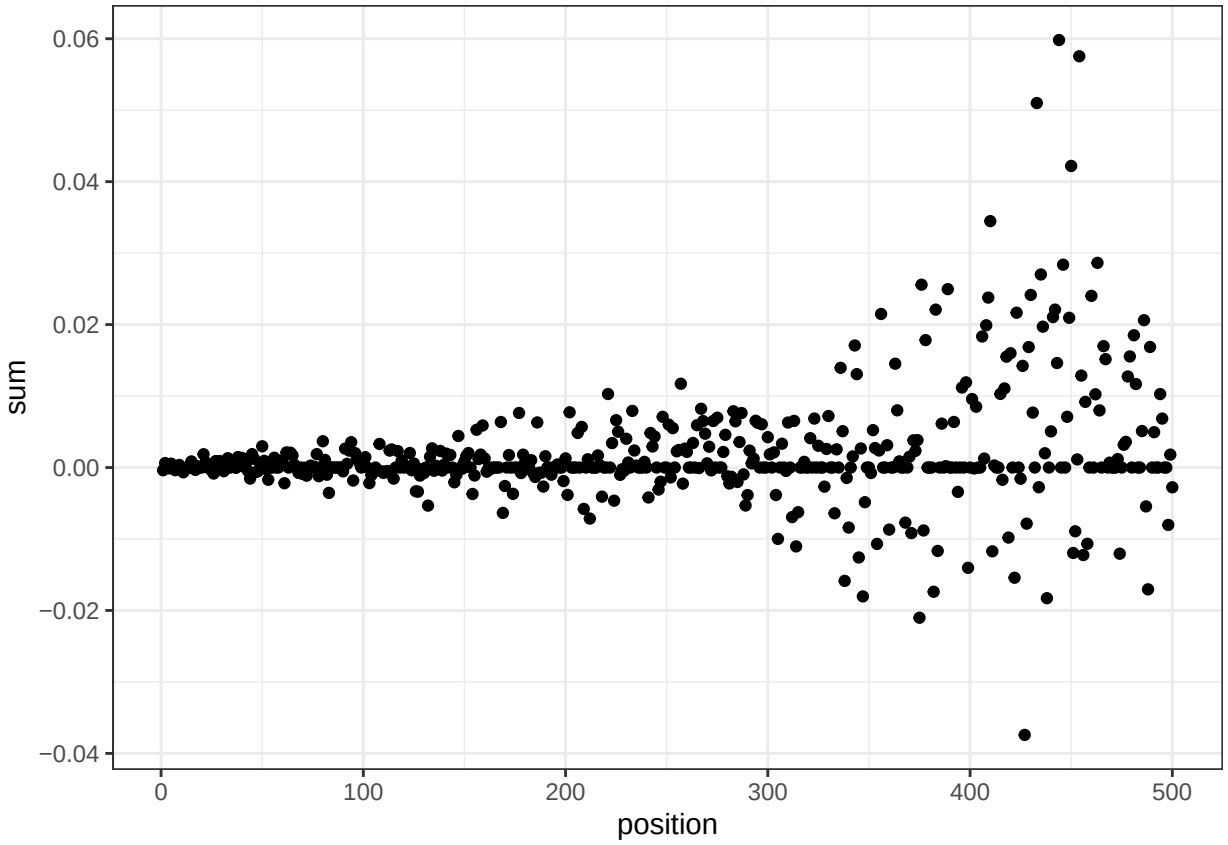
```
##           [,1]        [,2]
## [1,] 0.9987847 0.001215239
```

```
## [[1]]
```




```
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```





We noticed approximately 0 importance assigned to the first 300 bp, and a monotonic rise in the last 200.

We repeat the process on different targets.

We discovered that the emphasis on the tail is recurrent whatever instance, baseline, or sequence starting index. This does not reflect the true traits of 16S rRNA gene, and is not logical.

We noticed the model specifications:

```
model <- create_model_lstm_cnn(
  maxlen = 500, # not divisible by 3
  layer_lstm = NULL,
  layer_dense = c(2L),
  vocabulary_size = 4,
  kernel_size = c(12, 12, 12),
  filters = c(32, 64, 128),
  pool_size = c(3, 3, 3),
  learning_rate = 0.001)

train_model(train_type = "label_folder",
  model = model,
  path = c(path_16S_train, path_bacteria_train),
  path_val = c(path_16S_validation, path_bacteria_validation),
  vocabulary_label = c("16s", "bacteria"),
  path_checkpoint = checkpoint_path,
  train_val_ratio = 0.2,
  run_name = run_name,
  batch_size = 256,
  steps_per_epoch = 25,
  epochs = 8,
  save_best_only = FALSE,
  step = c(100, 500), # take sample every 100 step for 16S and every 500 for bacteria
  proportion_per_seq = c(0.95, 0.05))
```

The length is not divisible by 3, and when the instance is introduced after `substr`, the original codon structure is broken, and the model did very likely not capture codon structure during the training at all.

We speculate the model to be not biologically trustworthy. We thus retrain a model.

```
model <- create_model_lstm_cnn(
  maxlen = 600, # size divisible by 3
  ...)

train_model(train_type = "label_folder",
  ...
  step = c(6, 30), # smaller steps, divisible by 3
  ...)
```

Retrained Model

We load the model at last epoch:

```
## Using checkpoint checkpoints/16S_vs_bacteria_full_2/Ep.008-val_loss0.13-val_acc0.991.hdf5

## [[1]]
```

```
## [[1]]$confusion_matrix
##           Truth
## Prediction  16s bacteria
##   16s      1198        1
##   bacteria   52     1249
##
## [[1]]$accuracy
## [1] 0.9788
##
## [[1]]$categorical_crossentropy_loss
## [1] 0.1410726
##
## [[1]]$AUC
## [1] 0.9999059
##
## [[1]]$AUPRC
## NULL
```

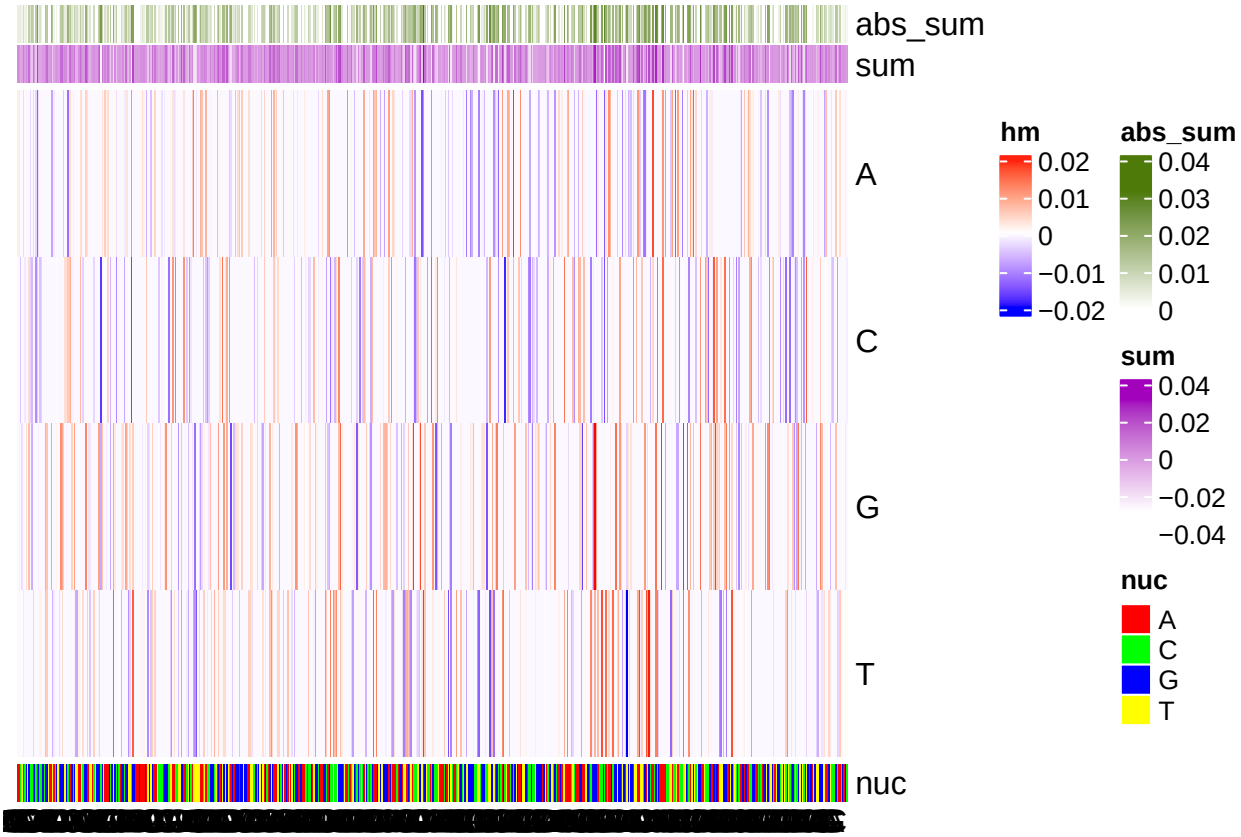
It still has a very high accuracy. We evaluate it again using modified IG.

Instance: GCF_001986655.1_ASM198665v1_genomic.16s.fasta 499-1098 bp.

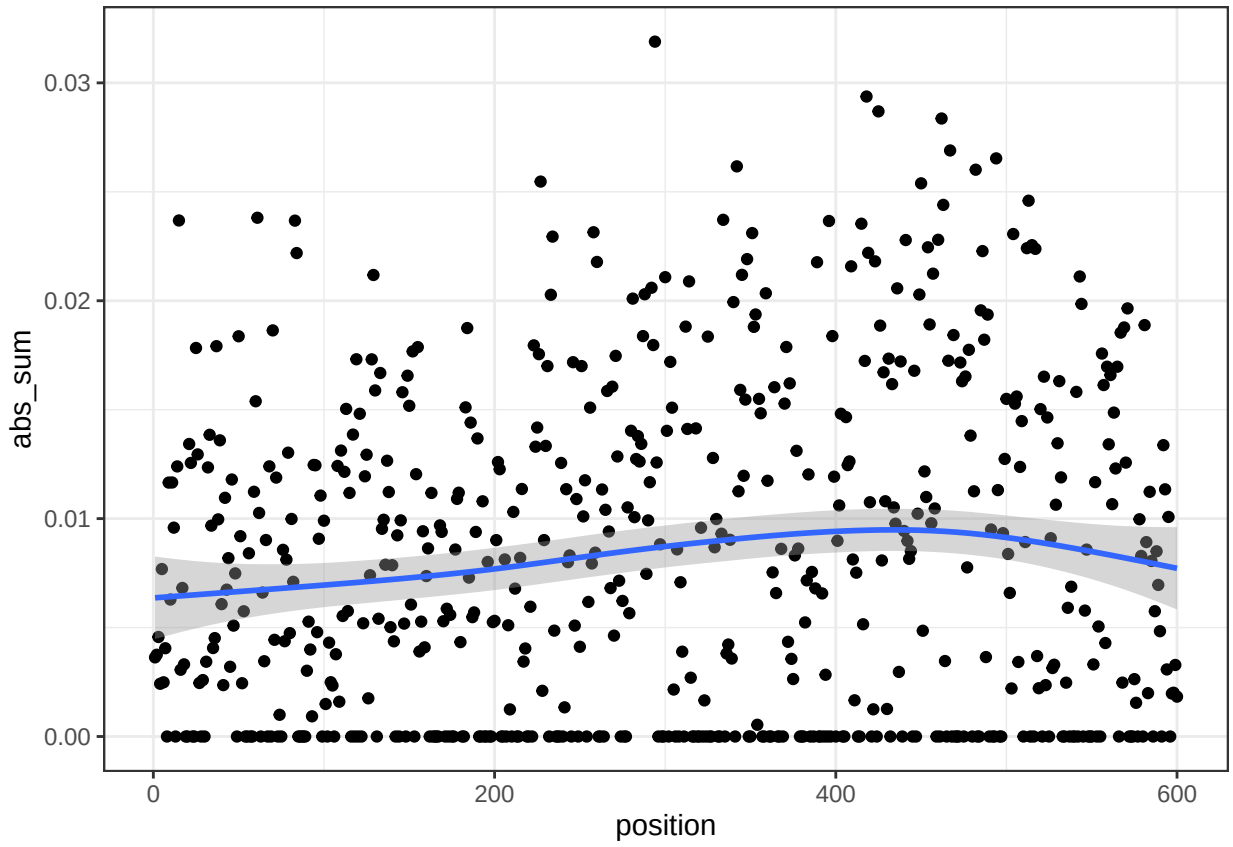
Baseline: GCF_002895085.1_ASM289508v1_genomic.fasta 499-1098 bp.

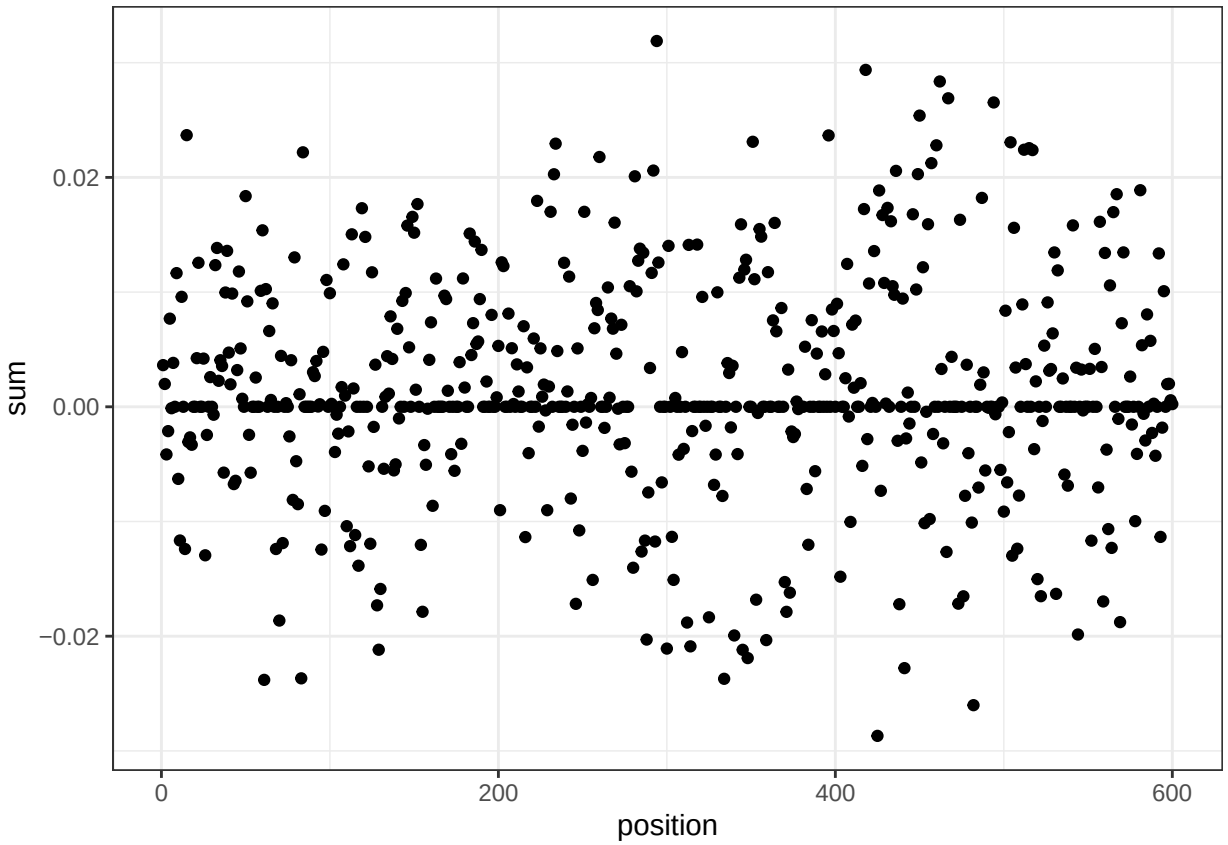
```
##           [,1]       [,2]
## [1,] 0.906086 0.09391395
```

```
## [[1]]
```



```
## `geom_smooth()` using formula = 'y ~ s(x, bs = "cs")'
```



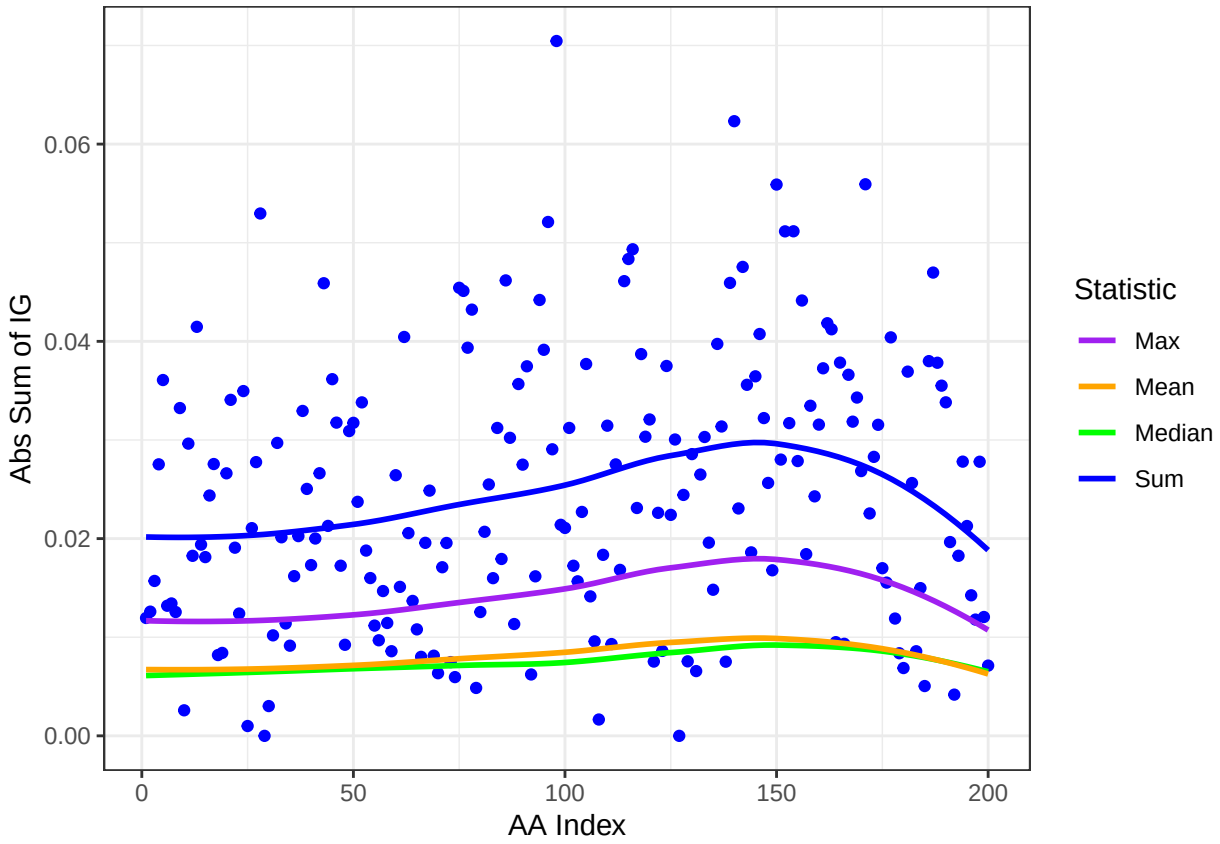


This time, the importance is scattered across positions in a more complex way, which reflects the nature of genetic data.

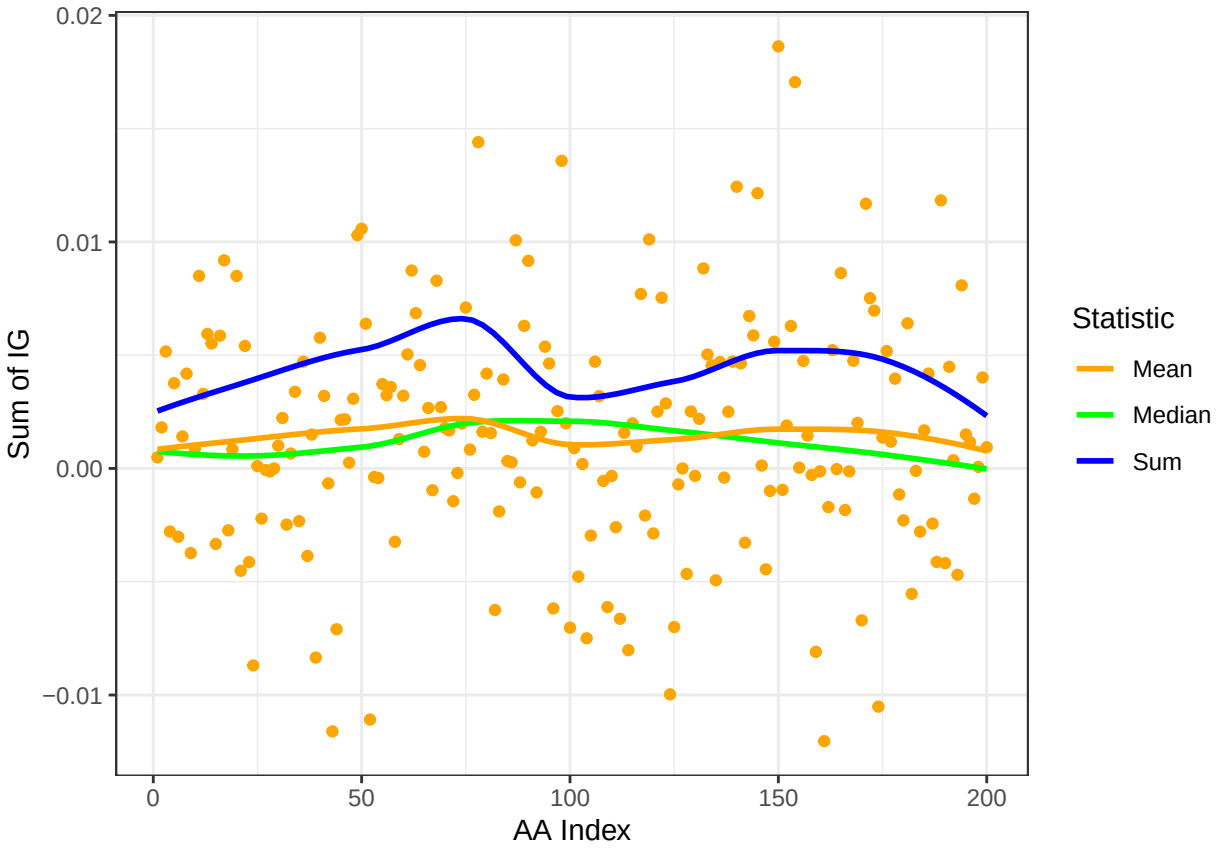
We also visualize the direct sum, since some points persistently have negative gradients. The implication of these is still unknown.

The same information compression technique is shown below.

```
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```

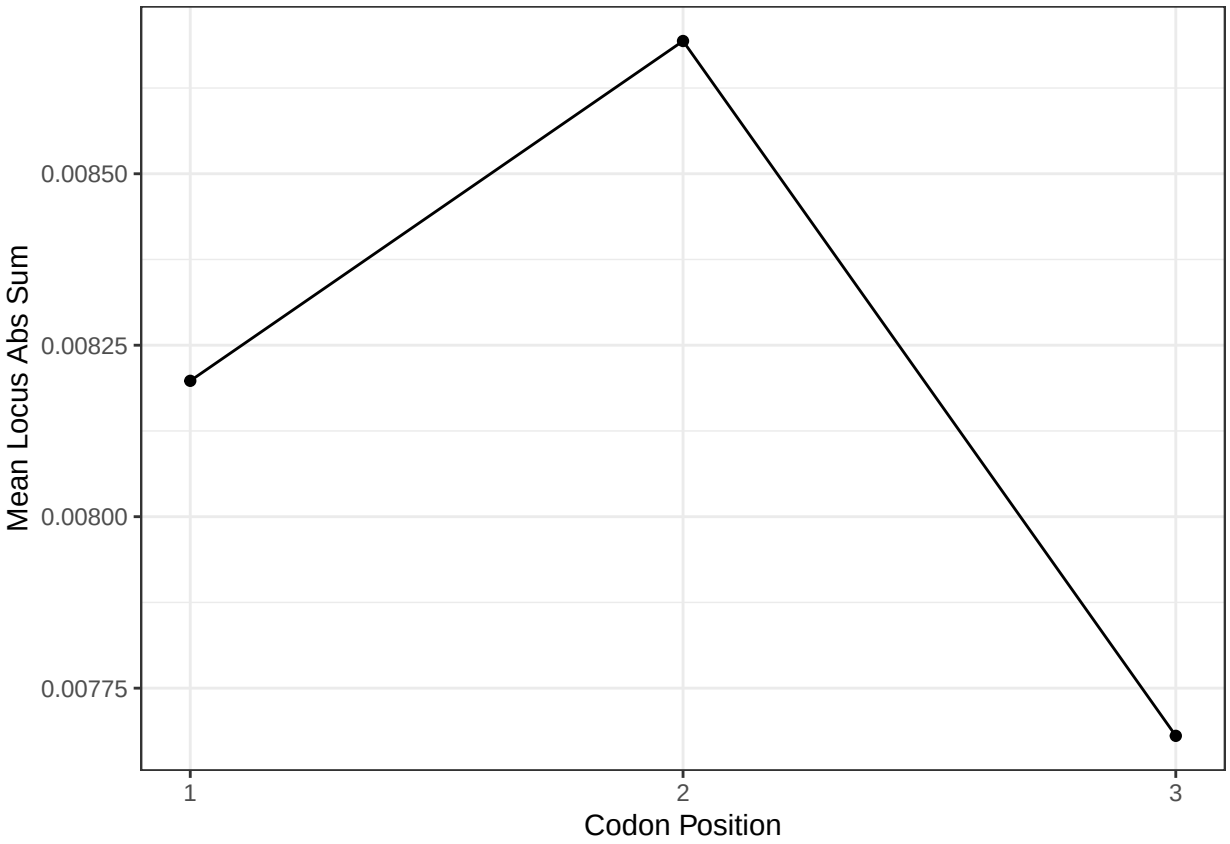


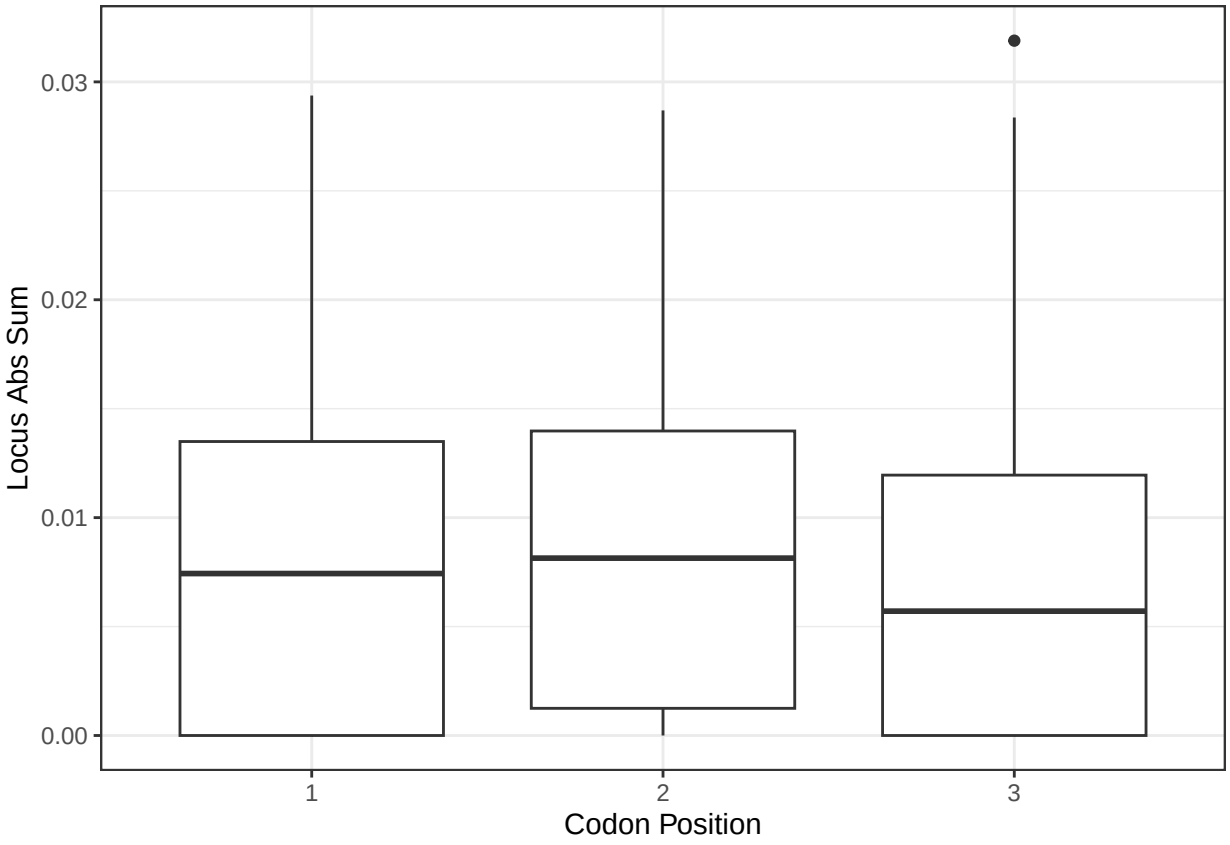

```
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```



We further studied wobleness (see Durrant & Bhatt). In this case, more randomness is introduced by the fact that we only calculated IG once. In RSDexport, the average importance almost consistently follow the 2-1-3 structure. Here, we tested some other instances and baselines and wobleness is not always detected.

Here is an example of existing wobbleness.


Mean Locus Abs Sum
0.00850
0.00825
0.00800
0.00775
1
2
3
Codon Position

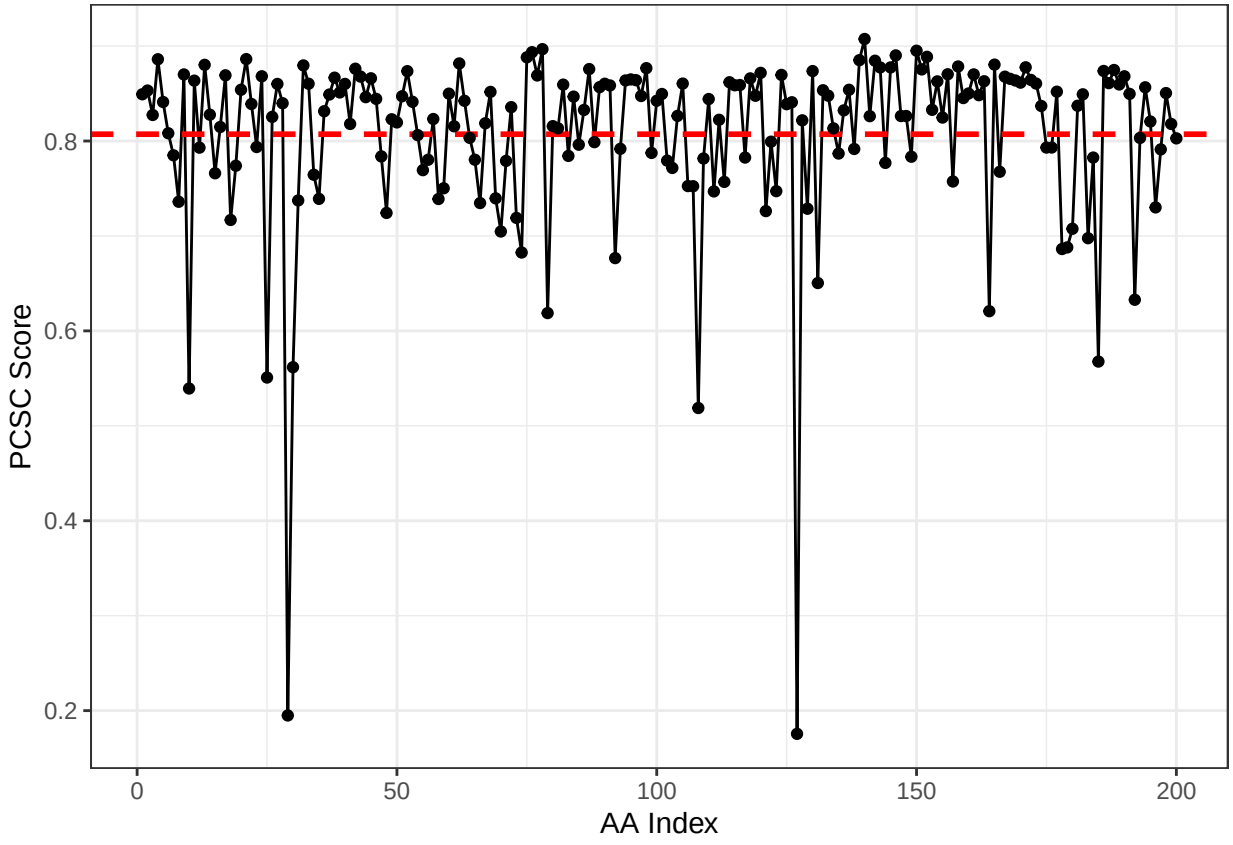

Locus Abs Sum
0.03
0.02
0.01
0.00
1
2
3
Codon Position

```
##
## Call:
## lm(formula = abs_sum ~ factor(position), data = codon_data)
##
## Residuals:
##        Min         1Q     Median         3Q        Max
## -0.0086935 -0.0076805 -0.0007748  0.0049028  0.0242104
##
## Coefficients:
##                      Estimate Std. Error t value Pr(>|t|)
## (Intercept)         0.0081981  0.0005280  15.527   <2e-16 ***
## factor(position)2   0.0004955  0.0007467   0.664    0.507
## factor(position)3 -0.0005176  0.0007467  -0.693    0.488
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.007467 on 597 degrees of freedom
## Multiple R-squared:  0.003075,   Adjusted R-squared:  -0.0002652
## F-statistic: 0.9206 on 2 and 597 DF,  p-value: 0.3989
```

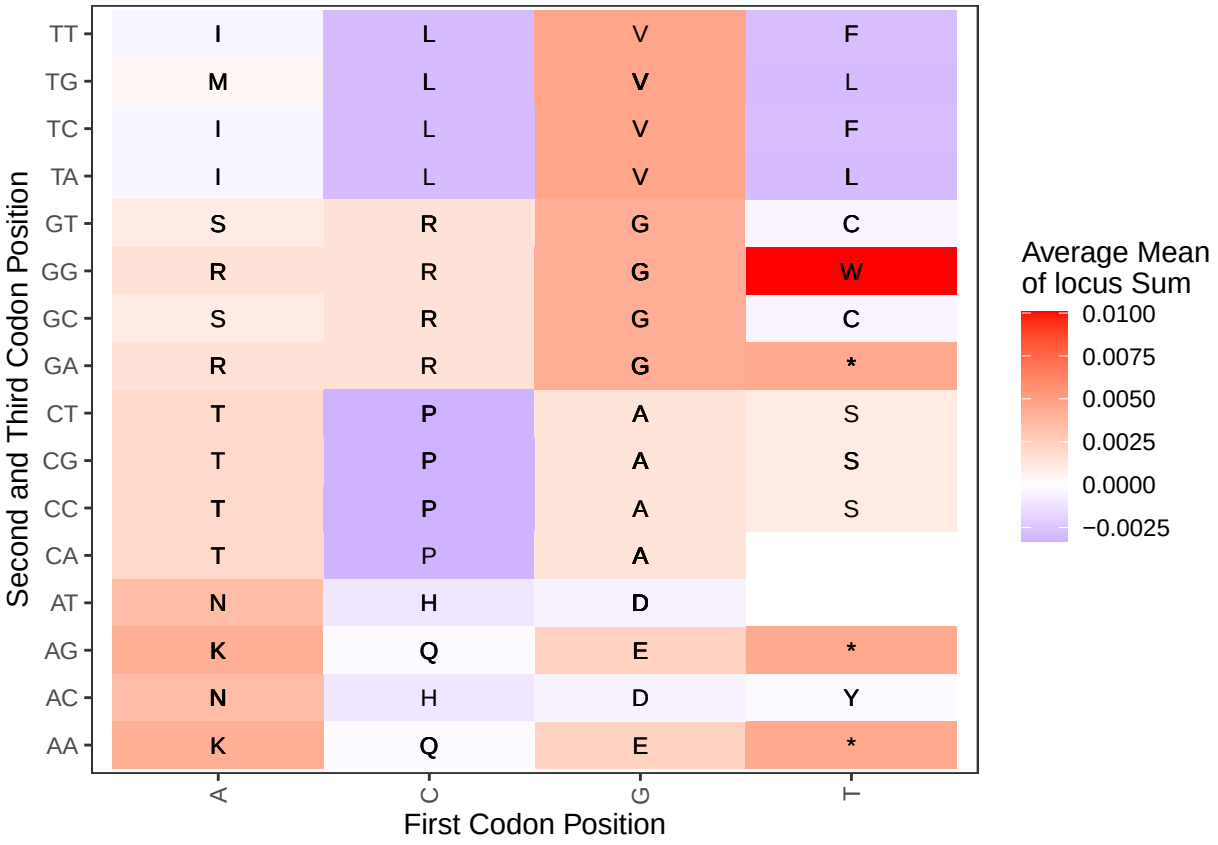
Consistency

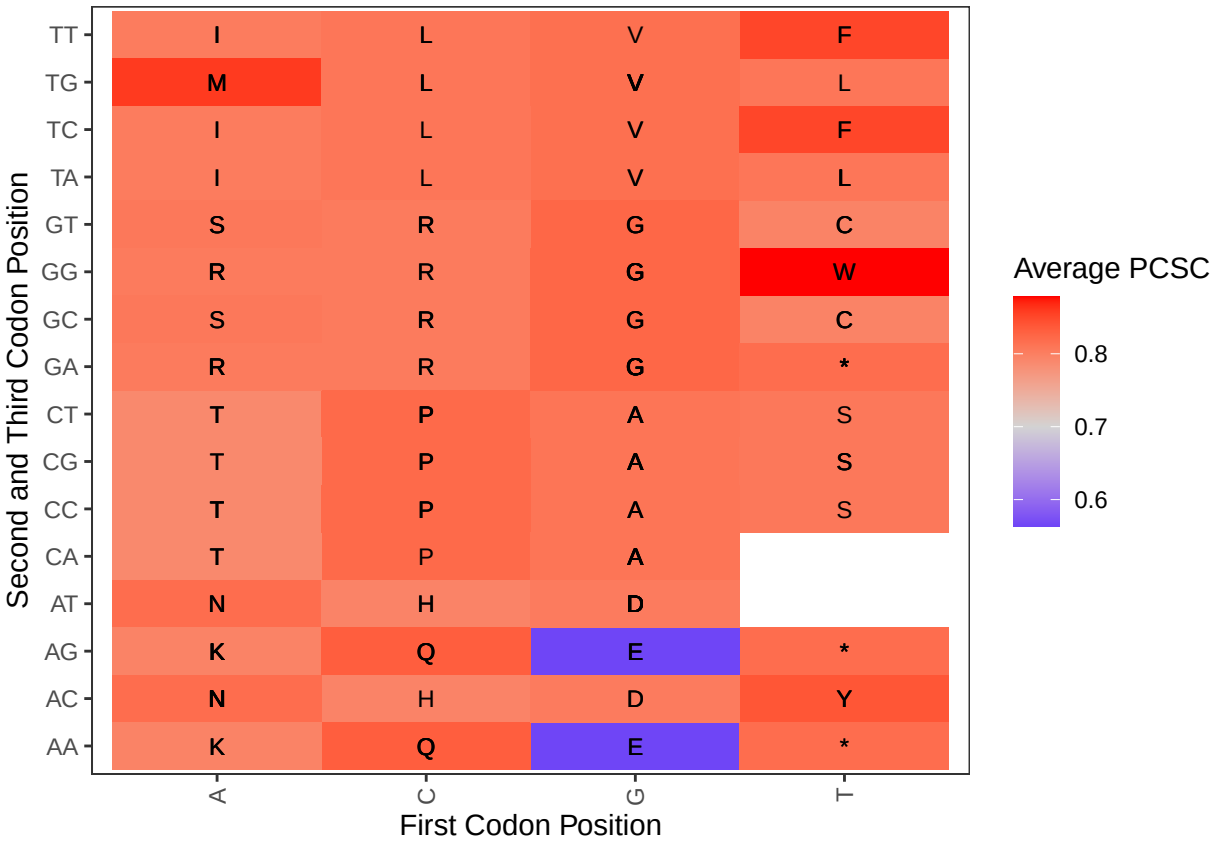
We then calculated PCSC score on our data. With an average score at 0.8, we conclude that the model was able to capture synonymous codons and treats them rather consistently.

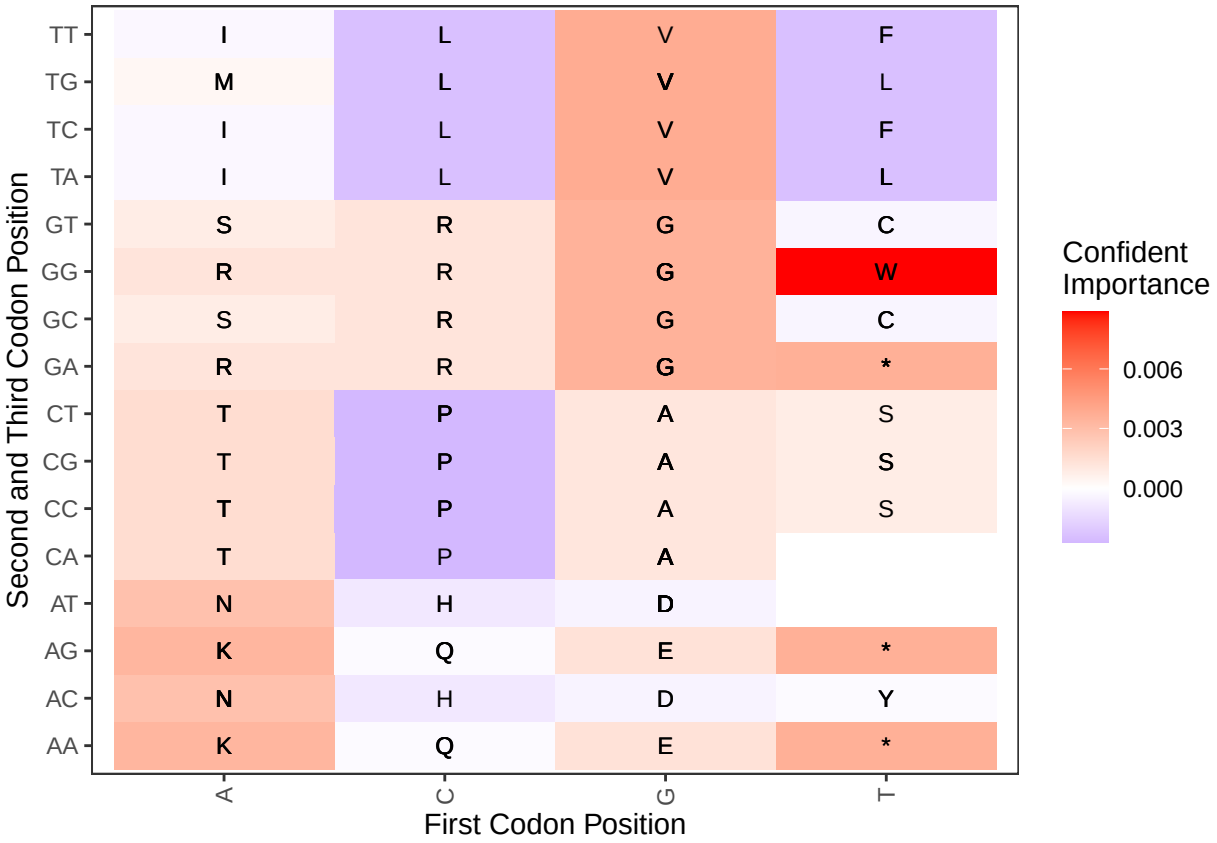
```
## Loaded result_df from existing CSV file.
```



We then generated heatmaps as in RSDexport. The first heatmap visualizes average sum of gradients, the second average consistency based on AA, and the third the "confident importance", which is average sum of gradients multiplied by PCSC score.




Second and Third Codon Position
TT I L V F
TG M L V L
TC I L V F
TA I L V L
GT S R G C
GG R R G W
GC S R G C
GA R R G *
CT T P A S
CG T P A S
CC T P A S
CA T P A
AT N H D
AG K Q E *
AC N H D Y
AA K Q E *
A C G T
First Codon Position
Average PCSC
0.8
0.7
0.6


Second and Third Codon Position
TT I L V F
TG M L V L
TC I L V F
TA I L V L
GT S R G C
GG R R G W
GC S R G C
GA R R G *
CT T P A S
CG T P A S
CC T P A S
CA T P A
AT N H D
AG K Q E *
AC N H D Y
AA K Q E *
A C G T
First Codon Position
Confident Importance
0.006
0.003
0.000

Feature Selection

We then implemented the feature selection algorithm as by the enhanced IG paper. We do this based on locus level, and sample 50 events for both interest and random group.

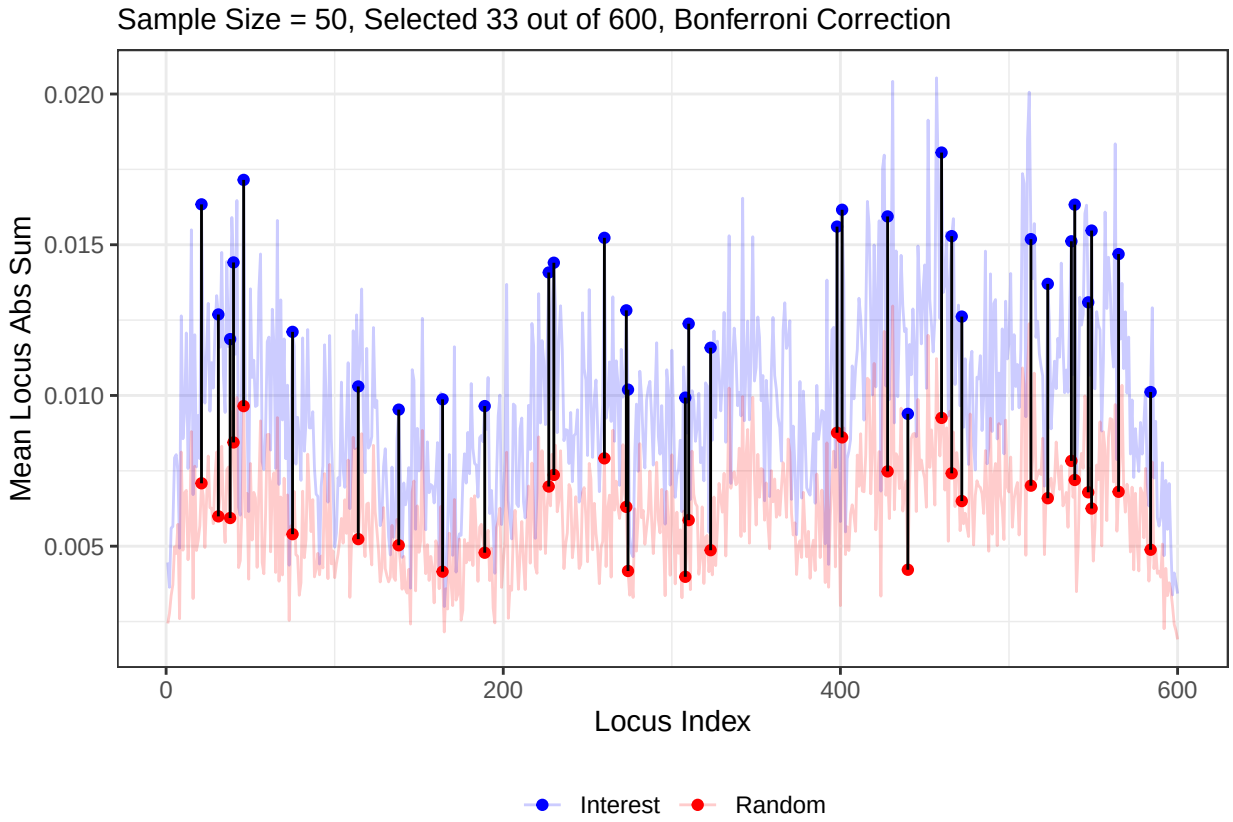
We performed one-sided t-test row-wise (only select positions with absolute sum of gradients larger than the random group). We applied Bonferroni Correction. (/600)

In the end, 33 out of 600 loci are identified as having significantly higher absolute sum of gradients than the random group.

We visualize these pairs, and notice that high importance points are not necessarily helpful in distinguishing instances, since the two curves are very similar in shape.

```
## Loaded interest_df from existing CSV file.
```

```
## Loaded random_df from existing CSV file.
```



We further explore the consitution of those important features. We only extracted the indices, and we compare them with one of the instances and extract the AA position of those points.

At the moment, **more A and G seem to be present.**

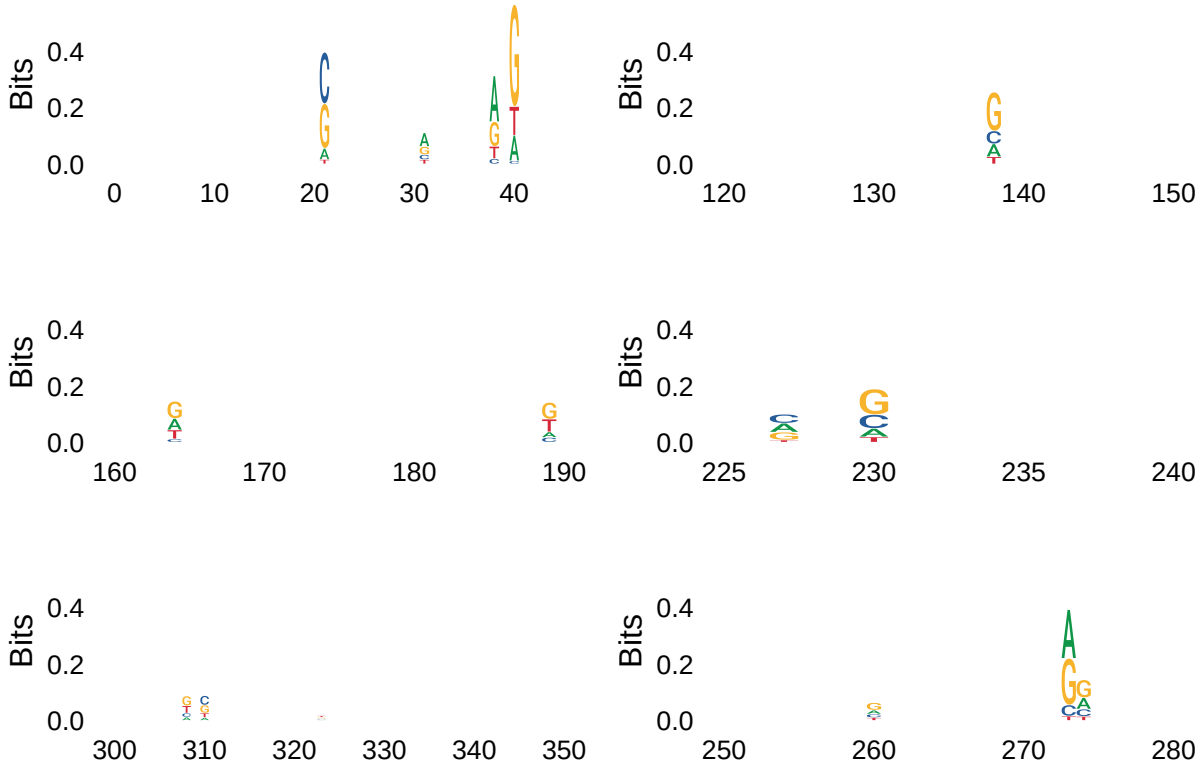
We noticed some consecutive pairs – loci that are next to each other and are all significant in test. They are extracted for possible study.

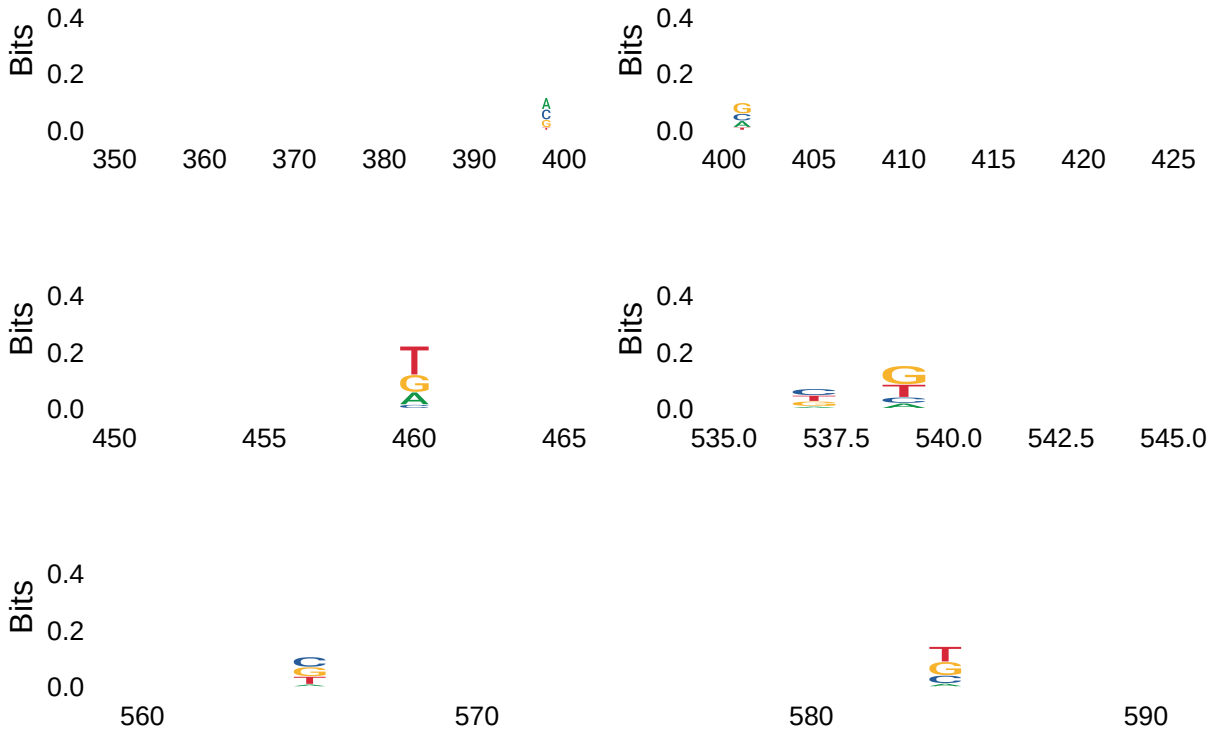
```
## [1] "ACGT content of selected features:"
```

```
## selected_trip
##  A  C  G  T
## 39 21 22 17
```

```
## [1] "Table of corresponding amino acids of selected features:"

## selected_key
## * A D G H I K L N Q R S T V Y
## 1 4 2 1 2 1 4 3 2 4 1 1 4 2 1

##   position    row_mean random_mean      p_value significance AApos trip key
## 1       38 0.011869143 0.005934357 8.528775e-06            *    13  CAA   Q
## 2       40 0.014412973 0.008441982 3.425646e-05            *    14  GCG   A
## 3      230 0.014403446 0.007363521 1.147962e-05            *    77  TCC   S
## 4      308 0.009927865 0.003984371 1.316175e-06            *   103  AAC   N
## 5      310 0.012380329 0.005864532 2.832893e-05            *   104  GAT   D

## [1] "ACGT content of selected, consecutive features:"

## consecutive_trip
## A C G T
## 5 5 3 2

## [1] "Table of corresponding amino acids of selected, consecutive features:"

##
## A D N Q S
## 1 1 1 1 1
```

We have visualized the selected features in **a sequence logo plot**. Keep in mind that here, G and A are present with relatively high confidence.





Adversarial

We also tried adversarial instances. We replace the selected spots of the instance all with C, and the confidence for 16S rRNA drops dramatically! Also replacing them with T reduces the confidence to 0.20. Replacing with A and G is not effective.

We compare them with an exact same number of randomly selected positions and substitute them accordingly too.

We also tried replacing them with 0.25, and the confidence drops only a little bit. There is also only little difference of confidence between the selected features and randomly sampled features.

```
## [1] "Original Prediction: "

##           [,1]       [,2]
## [1,] 0.906086 0.09391395

## [1] "Prediction after substituting important features with C: "

##            [,1]      [,2]
## [1,] 0.1646671 0.8353329

## [1] "Prediction after substituting same amount of random features with C: "

## [1] 0.3719506

## [1] "Prediction after substituting important features with A: "
```

```
##           [,1]       [,2]
## [1,] 0.9320796 0.06792048
```

```
## [1] "Prediction after substituting same amount of random features with A: "
```

```
## [1] 0.9210785
```

```
## [1] "Prediction after substituting important features with 0.25:"
```

```
##           [,1]      [,2]
## [1,] 0.7860437 0.2139563
```

```
## [1] "Prediction after substituting same amount of random features with 0.25:"
```

```
## [1] 0.8143192
```

We would like to study the drastic drop of prediction confidence when more C and T are introduced.

Input Reconstruction

We tried reconstructing the input sequence using a non-informative baseline, where A,C,G,T each has 0.1 for each position. We attempted 0.25, but the algorithm did not converge. Also choosing a bacteria sub-sequence leads to a non-converging result. This seems to be interesting.

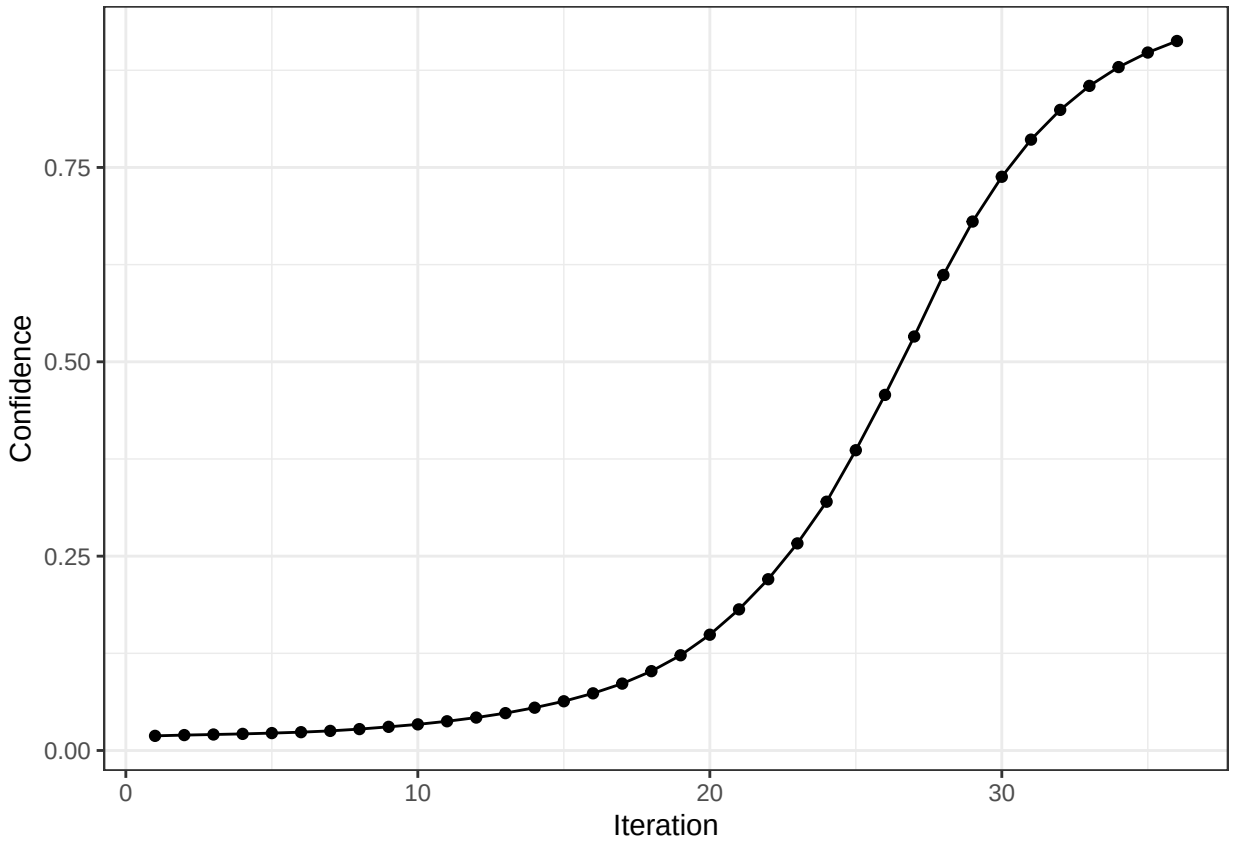
This is done by iteratively doing:

$$X^{(t+1)} = X^{(t)} + \epsilon \cdot IG(X^{(t)})$$

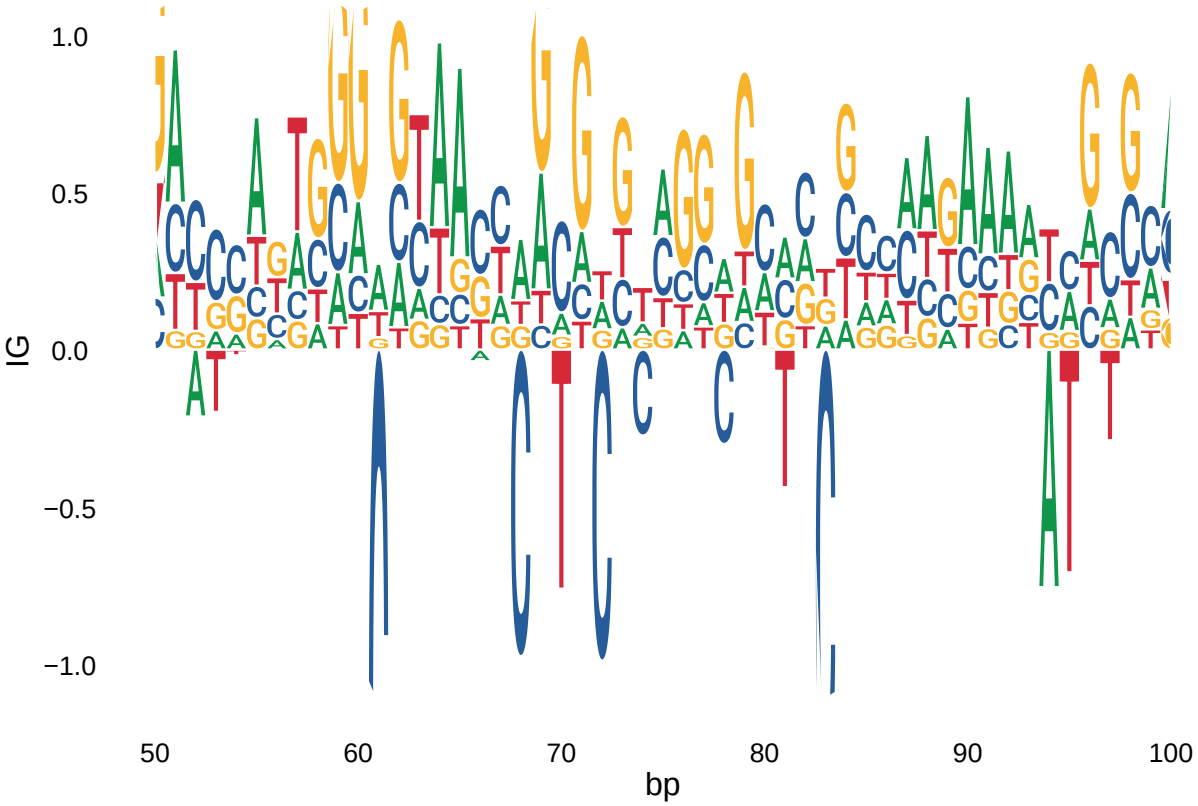
We set $\epsilon = 2$.

We plot the learning curve and the sequence logo (customed using final IG as y-axis) for a segment of the sequence.

Interpretation: the achieved representation is the least required to be predicted as 16S rRNA gene with a confidence of 90%.




```
## Scale for x is already present.
## Adding another scale for x, which will replace the existing scale.
```



Pay extra attention to the reconstructed sequence logo plot. In the negative region, C and T often have considerably negative weights, meaning they should not appear on this position if the prediction is 16S. This could explain why the adversarial scenario where more Cs are included caused dramatic drop in confidence.

We also table the matching positions of the reconstructed sequence comparing the original one. Compared to selected features, we see the matching positions seem not to be totally away from the selected ones. Could this be of interest?

```
## [1] "Matched Reconstructed sequence and original sequence, AA position: "
```

```
## AAT GTG GTA CGG GTG CAA GGA GGG GGA AGA AAA AAC AGG AAT AAT TAT GTG GAA TAG TGA
##   1   3   8  11  12  13  17  20  26  30  31  34  36  41  45  46  50  56  60  62
## GGA GGA TGG GAC AAT ACC AAG GGG GAG ACC GAC GTA GCC GAC GGG AGT AAT AGC GGG GGC
##  68  69  78  79  80  81  87  89  90  92  95  98 101 104 113 116 119 122 126 129
## ACG TGA GGC GCG GTG GTG GAC GAC CTC GAT GGA CGC AGC AAG AGA ACA GGA GGG CTA AGT
## 134 138 140 144 145 148 149 152 161 162 165 169 172 175 176 177 178 194 195 196
```

```
## [1] "Selected features, AA position: "
```

```
##  [1]   7  11  13  14  16  25  38  46  55  63  76  77  87  91  92 103 104 108 133
## [20] 134 143 147 154 156 158 171 175 179 180 183 183 189 195
```

Discussion

Main Point: **IML in Genomics: What comes after non-informative local interpretations? The Interpretation of Interpretations**

1. Due to lack of clear patterns in most real-world cases, it seems IG alone, even if correctly calculated, can be very much not informative. Instead of keep improving its theory minimally, it seems more reasonable to come up with methods tailored to genomics that can best utilize those data and provide interpretable or meaningful results. For example, the PCSC score could be a good measurement in explaining the reliability of the model in capturing synonymous codons.

2. How to work with the selected features? Are they biologically significant? Are they useful in distinguishing instances? (for which we need to adjust the model) These are questions that need to be answered in the future.

3. I personally find the convergence of feature reconstruction fascinating. For some reasonable targets, e.g. a bacterial sequence, it is not possible to transform it to a possible 16S simply using IG outputs.

4. The choice of baseline can be further studied. Choosing a real sequence from the other category can include too much artifact than truth.