COVID-19 mRNA Vaccine Degradation Prediction

Group 20
陳英翔　F64099552
王俊霖　F74066111
鄭宇澤　F74067094

一、問題簡介

mRNA 疫苗已率先成為 COVID-19 開發最快的候選疫苗，但目前它們面臨著潛在的限制。最大的挑戰之一是如何提高 RNA 分子的穩定性，由於具有自發分解的趨勢，因此需要在零下好幾十度的環境下保存。傳統疫苗能夠在冷藏條件下運送到世界各地，但目前 mRNA 疫苗無法做到這一點。

而我們要探討的問題是，如何去預測 mRNA 中的鹼基在不同環境不同位置下的降解程度。如果缺少這些數值，很難去改善 mRNA 疫苗在保存上的穩定性。

二、資料集說明

I. 資料來源 https://www.kaggle.com/c/stanford-covid-vaccine/data

資料集來自 kaggle 平台，是由史丹佛大學的醫學院所提供。 預測結果可以透過該平台來做評分。

II. 資料集內容

	特徵名稱	說明	資料型態
Input X	id	每段 mRNA 的個別 id	字串
	seq_length	mRNA 的鹼基長度(107、130 兩種)	數值
	sequence	鹼基序列(A/U/C/G)	字串
	structure	使用()和.來表示鹼基是否配對	字串
	error	表示 deg_列和 reactivity 的出錯率	數值
	predicted_loop_type	鹼基的構造類別(S/M/I/B/H/E/X)	字串
Output Y	reactivity	各個鹼基的反應性	數值
	deg_pH10	pH10 之下各個鹼基的降解率	數值
	deg_Mg_pH10	pH10 且含鎂的環境下各個鹼基的降解率	數值
	deg_50C	50 度的環境下各個鹼基的降解率	數值
	deg_Mg_50C	含鎂且 50 度的環境下各個鹼基的降解率	數值

Ouput Y 中有五項預測值，分別是 mRNA 中的所有鹼基在不同環境下的降解度，需要透過 Input X 中的序列、構造、位置等特徵來求得。

1. 資料集大小(mRNA 個數)

   每一筆資料皆為一個個別的 mRNA。

   train.json ： 2400 筆

   test.json ： 3634 筆

2. 資料集大小(鹼基個數)

   (1)train data

```
107    2400
Name: seq_length, dtype: int64
```

   特徵 seq_length 代表 mRNA 鹼基長度，只存在 107、130 這兩種值。根據觀察 train data 的結果，train data 當中所有 2400 筆資料的 mRNA 長度皆為 107。而主辦方由於技術上的問題沒有辦法提供完整 107 個鹼基的數據，每個 mRNA 只提供了前 68 個鹼基的數據。 因此「68 個鹼基 X 2400 筆資料」總共會有 163200 個鹼基的數據被拿來做訓練。

   (2)test data

```
130    3005
107     629
Name: seq_length, dtype: int64
```

   根據觀察 test data 的結果，與 train data 的不同點在於 mRNA 鹼基長度種類不只一種，其中 3005 筆資料的 mRNA 長度為 130、629 筆資料的 mRNA 長度為 107。 主辦方要求對 test data 進行預測的時候，必須是要針對所有的鹼基，因此「130 個鹼基 X 3005 筆資料」加上「107 個鹼基 X 629 筆資料」總共需要預測出 457953 個鹼基的降解率。

3. 評分方式

$$\text{MCRMSE} = \frac{1}{N_t}\sum_{j=1}^{N_t}\sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_{ij}-\hat{y_{ij}})^2}$$

   我們將使用 kaggle 平台來對預測結果做評分，評分方式是使用 RMSE(均方根誤差)。 由於會預測出多個 RMSE 值，因此需透過取平均值得到最終評估指標。 評分對象為 Ouput Y 當中的「reactivity、deg_Mg_pH10、deg_Mg_50C」這三項。

   Kaggle 平台會顯示兩種數據分別是 Public Score 和 Private Score，Public Score 是使用 9%的測試數據計算得出的分數，Private Score 則是使用 91%的測試數據所計算出來的分數。

三、資料分析與處理

I. 資料視覺化

1. DataFrame 形式

將資料轉換為 DataFrame 形式後，可看出 Input X 中每一個 row 代表一個個別的 mRNA，由字串所組合。 Ouput Y 皆為連續型數值，每一個 row 代表的是一個鹼基，是將 Ouput X 的每個 mRNA 拆散後的結果。

Input X

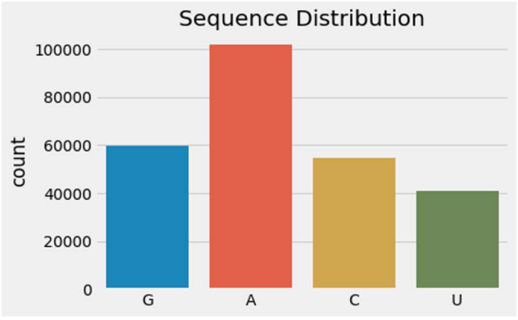
	sequence	structure	predicted_loop_type
	GGAAAAGCUCUAAUAACAGGAGACUAGGACUACGUAUUUCUAGGUA...	.....((((((.....))))).)).((.....((.. ((((((.....	EEEEESSSSSSHHHHHHHSSSSBSSXSSIIIIISSIISSSSSSHHH...
	GGAAAAAGCGCGCGCGGUUAGCGCGCGCUUUUGCGCGCGCUGUACC...	..... ((((((((((((((((((((....)))))))))).)))...	EEEEESSSSSSSSSSSSSSSSSSSSSSSSSSHHHHSSSSSSSSSSBSSS...
	GGAAAGUGCUCAGAUAAGCUAAGCUCGAAUAGCAAUCGAAUAGAAU...	.....((((.((.....((((.(((.....))))).. ((((.....)...	EEEEESSSSISSIIIIISSSSMSSSHHHHHSSSMMSSSSHHHHHHS...

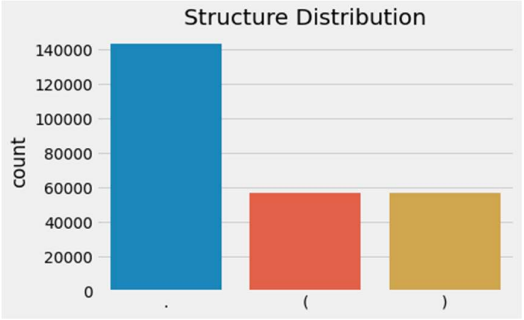
Output Y

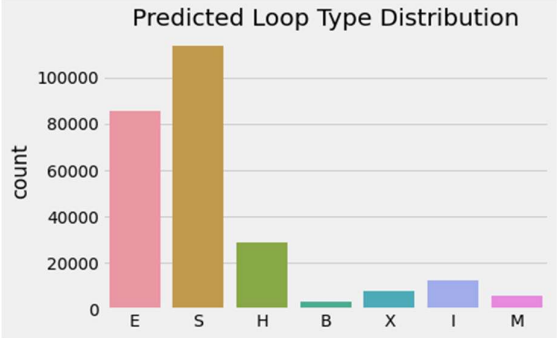
reactivity	deg_Mg_pH10	deg_pH10	deg_Mg_50C	deg_50C
0.703995	0.657001	2.049222	0.486849	0.657529
2.455442	3.572951	4.111922	3.209466	2.386644
1.391439	0.700591	0.956595	0.746343	0.945210
1.161069	0.947396	1.100488	1.416168	1.157993
0.635671	0.483556	0.642555	0.799200	0.755601

2. 數量統計

針對特徵作視覺化，先將 Structure、Sequence、Loop Type 做數量上的統計，但因為是序列資料以及數值預測，無法直接從這三個特徵的數量做出初步推論。



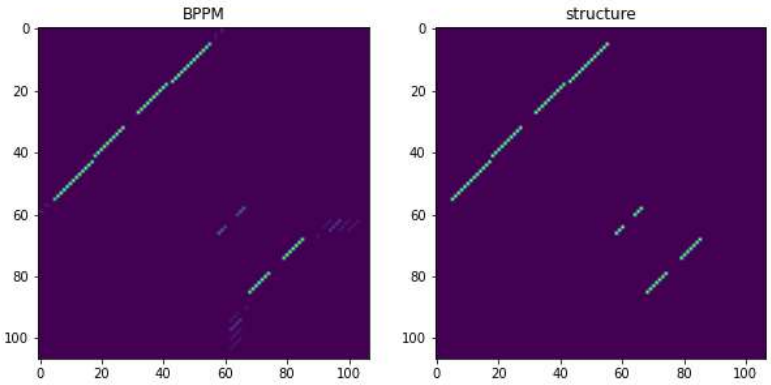




II. 特徵工程

1. BPPM

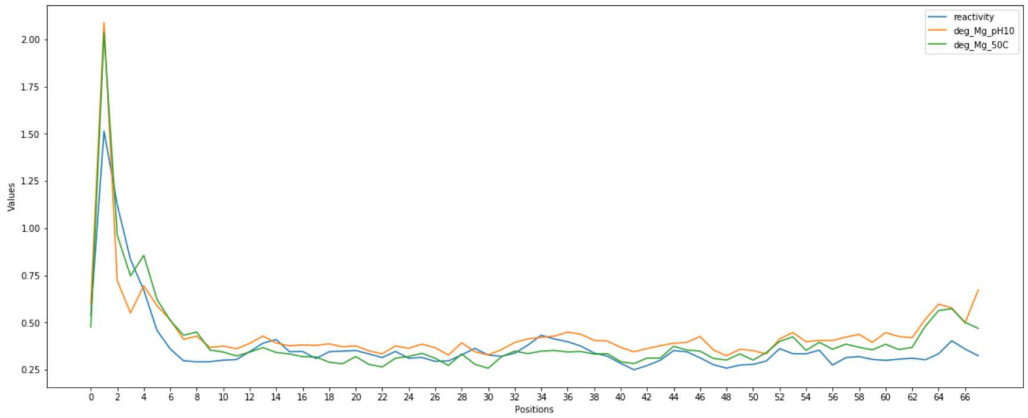
在原始資料集中，主辦方附帶了名叫 BPPM 的資料夾，檔案數量跟序列總數相同，我們猜想這應該是每個 mRNA 都有的特徵，但確切代表意義過於學術，主辦方也沒特別多講，因此在資訊不夠的情況下無法分析，於是決定上網找其他人的分析，發現繪製出來的分布圖和 Structure 這個特徵的分布很接近，推測是某種和配對與否有關係的變數，並且對鹼基的分解力、溶解率等數值預測有很大的影響，故決定放入變數之中，下圖為 training data 裡，BPPM 和 Structure 的分布比較：



嘗試放入 LightGBM 模型做預測，比較有加入 BPPM 以及未加入 BPPM 模型兩者的分數，發現加入了 BPPM 那組的 MCRMSE 小了 0.02，雖說看似相差不大，但這樣微小的差距卻讓我們在 kaggle 競賽中上升了 1、200 名，所以後續的模型都決定加入 BPPM。

2. 加入位置特徵

取得 train data 中所有 mRNA 序列資料的鹼基所在位置，如圖：



（上圖為我們平均了 train data 當中所有 mRNA 片段前 68 個鹼基的“降解和反應性”）

可以從圖中看出，在序列開始和結束的時候，“降解和反應性”有較高的值，故推斷鹼基的序列位置跟“降解和反應性”是有關係的，因此將其命名為 positions，並納入 train data 當中。

3. 鹼基、配對、結構擴充

加入三種特徵，第一個是四種鹼基(AUCG)分別在每段 mRNA 中所占的百分比，以前五筆 mRNA 序列資料為例：

sequence	A_percent	U_percent	C_percent	G_percent
GGAAAAGCUCUAAUAACAGGAGACUAGGACUACGUAUUUCUAGGUA...	0.420561	0.186916	0.214953	0.177570
GGAAAAAGCGCGCGCGGUUAGCGCGCGCUUUUGCGCGCGCUGUACC...	0.233645	0.158879	0.299065	0.308411
GGAAAGUGCUCAGAUAAGCUAAGCUCGAAUAGCAAUCGAAUAGAAU...	0.401869	0.186916	0.186916	0.224299
GGAAAAGCGCGCGCGCGCGCGCGAAAAAGCGCGCGCGCGCGCGCGC...	0.261682	0.084112	0.327103	0.327103
GGAAAAUAUAUAAUAUAUUAUAUAAAUAUAUUAUAGAAGUAUAAUA...	0.542056	0.345794	0.056075	0.056075

第二個則是加入序列中每種鹼基配對所佔的百分比，共有六種方式，分別為(A , U)、(U , A) 、 (G , C)、(C , G) 、 (U , G)、(G , U)，同樣以前五筆資料為例：

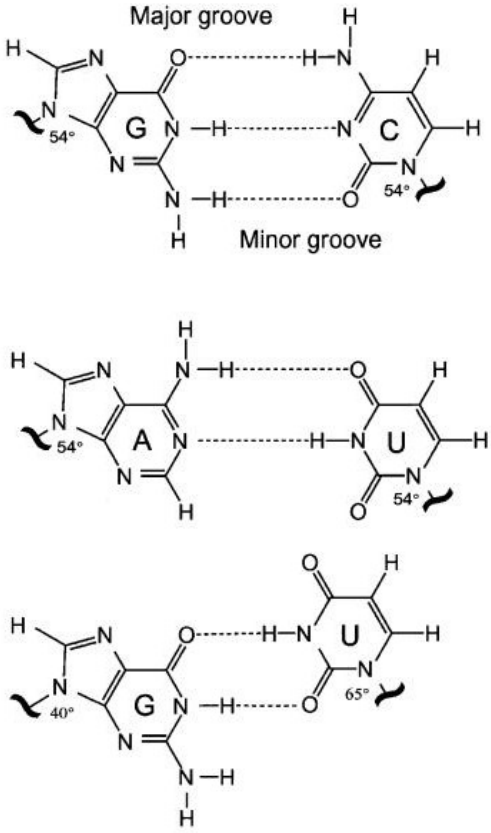
sequence	U-G	C-G	U-A	G-C	A-U	G-U
GGAAAAGCUCUAAUAACAGGAGACUAGGACUACGUAUUUCUAGGUA...	0.086957	0.130435	0.260870	0.347826	0.173913	0.000000
GGAAAAAGCGCGCGCGGUUAGCGCGCGCUUUUGCGCGCGCUGUACC...	0.030303	0.363636	0.030303	0.393939	0.121212	0.060606
GGAAAGUGCUCAGAUAAGCUAAGCUCGAAUAGCAAUCGAAUAGAAU...	0.041667	0.208333	0.208333	0.291667	0.125000	0.125000
GGAAAAGCGCGCGCGCGCGCGCGAAAAAGCGCGCGCGCGCGCGCGC...	0.000000	0.437500	0.062500	0.406250	0.031250	0.062500
GGAAAAUAUAUAAUAUAUUAUAUAAAUAUAUUAUAGAAGUAUAAUA...	0.000000	0.000000	0.457143	0.000000	0.542857	0.000000

第三個擴充為加入序列中每種結構所佔的百分比，共有七種結構，分別是 E、S、H、B、X、I、M，前五筆資料的狀況如圖：

predicted_loop_type	E	S	H	B	X	I	M
EEEEESSSSSSHHHHHHHSSSSBSSXSSIIIIISSIISSSSSSHHH...	0.242991	0.429907	0.140187	0.009346	0.046729	0.130841	0.000000
EEEEESSSSSSSSSSSSSSSSSSSSSSSSSHHHHSSSSSSSSSSBSSS...	0.242991	0.616822	0.102804	0.009346	0.000000	0.000000	0.028037
EEEEESSSSISSIIIIISSSSMSSSHHHHHSSSMMSSSSHHHHHHS...	0.242991	0.448598	0.140187	0.000000	0.009346	0.112150	0.046729
EEEEEESSSSSSSSSSSSSSSSSHHHHHHSSSSSSSSSSSSSSSSSS...	0.252336	0.598131	0.130841	0.018692	0.000000	0.000000	0.000000
EEEEESSSSSSSBSSSSSSSSSSSSSBSSSSSSSSSHHHHSSSSSSS...	0.242991	0.654206	0.074766	0.028037	0.000000	0.000000	0.000000

4. GC 含量

GC 含量這個詞是來自分子生物學的領域，GC 含量指的是，一個 mRNA 當中 GC 配對的鹼基所佔的含量。GC 含量愈高，會使【耐熱性、耐鹼性】提升，代表穩定性越高，不容易變質，可從下圖中看出，GC 含量就是為了要去強調這個氫鍵數量的不同，來表示穩定性。



圖中虛線代表的是氫鍵，可看出除了 GC 鹼基配對有 3 個氫鍵之外，另外兩種皆只有 2 個，因此 GC 可說是相較於其他兩種配對方式穩定的存在，故計算 GC 含量可了解此 mRNA 序列的穩定性。

由於這個競賽本身就是因為 RNA 分子具會有自發性分解的問題，所以要去預測鹼基在不同溫度、不同 PH 值之下的降解率，所以這項特徵很適合拿來放入模型中做預測，而 GC 含量的計算方式如圖：

$$\frac{G+C}{A+T+G+C} \times 100$$

III. 初步分析

使用由 RNN 衍生而來的變種模型 GRU，除了節省計算時間，也避免模型在做預測時，可能會遺失太遠之前的資訊，因此選擇相較於

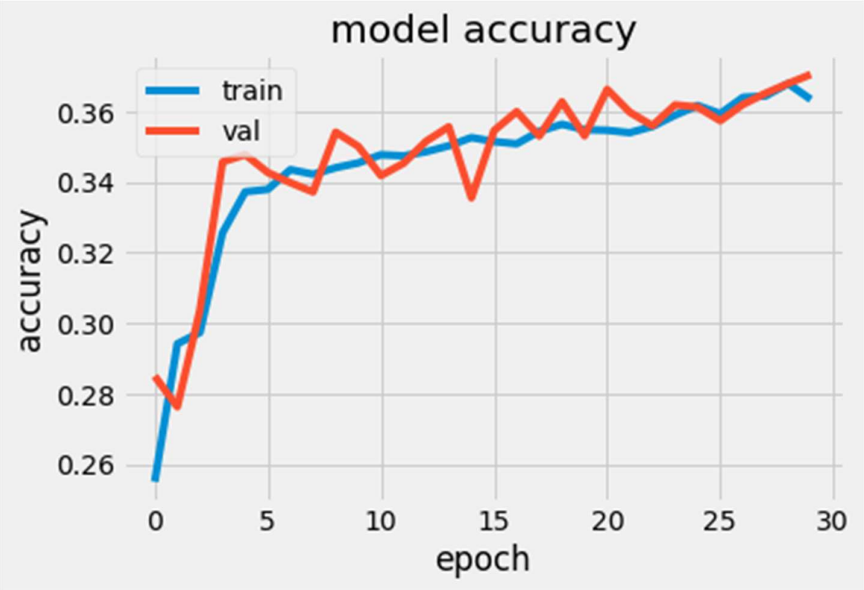
RNN，能記得更久之前的資訊的 GRU 模型，程式如下：

```
def build_model(embed_size,seq_len=107,sp_dropout=0.2,n_layers=3,
                hidden_dim=256,pred_len=68,dropout=0.5):
  inputs = L.Input(shape=(seq_len,3))

  embed=L.Embedding(input_dim=embed_size,output_dim=200)(inputs)
  reshaped=tf.reshape(
      embed, shape=(-1,embed.shape[1],embed.shape[2]*embed.shape[3])
  )
  hidden = L.SpatialDropout1D(sp_dropout)(reshaped)
  for x in range(n_layers):
    hidden = gru_layer(hidden_dim,dropout)(hidden)

  truncated = hidden[:,:pred_len]
  out = L.Dense(5,activation='linear')(truncated)
  model = tf.keras.Model(inputs=inputs,outputs=out)
  model.compile(tf.optimizers.Adam(),loss=MCRMSE,metrics=['accuracy'])

  return model
```

計算後的層數以及 Outputs 如圖：

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, 107, 3)]          0
_________________________________________________________________
embedding_1 (Embedding)      (None, 107, 3, 200)       2800
_________________________________________________________________
tf.reshape_1 (TFOpLambda)    (None, 107, 600)          0
_________________________________________________________________
spatial_dropout1d_1 (Spatial (None, 107, 600)          0
_________________________________________________________________
gru_3 (GRU)                  (None, 107, 256)          658944
_________________________________________________________________
gru_4 (GRU)                  (None, 107, 256)          394752
_________________________________________________________________
gru_5 (GRU)                  (None, 107, 256)          394752
_________________________________________________________________
tf.__operators__.getitem_1 ( (None, 68, 256)           0
_________________________________________________________________
dense_1 (Dense)              (None, 68, 5)             1285
=================================================================
Total params: 1,452,533
Trainable params: 1,452,533
Non-trainable params: 0
_________________________________________________________________
```

準確率計算出來後其實表現並不好，隨著迭帶次數越高，準確率並沒有因此提升許多，我們猜想可能不應使用 accuracy 當做評斷模型的標準，畢竟連續型的數值不容易被準確預測，故得到的結果最好也只有 0.4 左右，結果如圖：



四、模型訓練

使用 LGBM、CatBoost、NN 三種模型進行訓練。皆使用 grid search 來找出最佳參數組合。 會事先將資料集中每一筆 mRNA 資料拆散為鹼基資料。

I. 資料前處理

1. 原本的資料集中每一個 row 代表一個個別的 mRNA，資料為字串型式。此處要將每一筆資料給拆散，也就是將字串中的每一個字元給拆分出來，將一個 row 表示為一個鹼基。
2. 下一步將類別資料轉換為數字型態。

sequence	structure	predicted_loop_type
GGAAAAGCUCUAAUAACAGGAGACUAGGACUACGUAUUUCUAGGUA...	.....((((((.....)))))).)).((.....((.. ((((((.....	EEEEESSSSSSHHHHHHHSSSSBSSXSSIIIIISSIISSSSSSHHH...
GGAAAAAGCGCGCGCGGUUAGCGCGCGCUUUUGCGCGCGCUGUACC...	..... (((((((((((((((((((....)))))))))).)))...	EEEEESSSSSSSSSSSSSSSSSSSSSSSSSSHHHHSSSSSSSSSSBSSS...
GGAAAGUGCUCAGAUAAGCUAAGCUCGAAUAGCAAUCGAAUAGAAU...	.....((((.((.....((((.(((.....))))).. ((((.....)...	EEEEESSSSISSIIIIISSSSMSSSHHHHHSSSMMSSSSHHHHHHS...

【原本】 Train data : 2400rows

sequence	structure	predicted_loop_type
1	0	1
1	0	1
0	0	1
0	0	1
0	0	1

【拆散後】 Train data : 163200rows

II. 建構模型

Private Score、Public Score 的數據來自 kaggle 平台的評分結果。

1. LGBMRegressor

```
model = lgb.LGBMRegressor(num_leaves=101,
                          learning_rate=0.005,
                          feature_fraction=0.9,
                          bagging_fraction=0.8,
                          bagging_freq=5,
                          verbose=0,
                          early_stopping_rounds=100)
```

Private Score	Public Score
0.48761	0.42377

預測結果

2. CatBoostRegressor

```
model = CatBoostClassifier(depth=10,
                           iterations=1000,
                           l2_leaf_reg=5)
```

Private Score	Public Score
0.49204	0.42610

預測結果

3. NN

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
module_wrapper (ModuleWrappe (None, 500)               2500
_________________________________________________________________
batch_normalization_8 (Batch (None, 500)               2000
_________________________________________________________________
dropout_8 (Dropout)          (None, 500)               0
_________________________________________________________________
module_wrapper_1 (ModuleWrap (None, 50)                25050
_________________________________________________________________
batch_normalization_9 (Batch (None, 50)                200
_________________________________________________________________
dropout_9 (Dropout)          (None, 50)                0
_________________________________________________________________
module_wrapper_2 (ModuleWrap (None, 1)                 51
=================================================================
```

Private Score	Public Score
0.49052	0.42187

預測結果

五、RNN 模型訓練

因為是鹼基序列預測，應該將序列中的資訊也放入模型中，但在短時間內無法充分理解常用來處理序列資料的方法，例如 GRU、LSTM、Attention 等等，因此我們參考了他人的方法，並做理解與說明。

參考來源：https://www.kaggle.com/c7934597/covid-ae-pretrain-gnn-attn-cnn

使用的方法有 auto encoder pretrain model、GNN、Attn、CNN

I. 資料前處理

1. 篩選出 signal_to_noise > 1 的 mRNA
2. 對每筆 mRNA 的 Stucture 算出相鄰矩陣(adjacent matrix)。
3. 對每筆 mRNA 中的每個鹼基，根據該鹼基與其他鹼基之間的距離，算出另一個相鄰矩陣。
4. 對 Structure、Sequence、Loop Type 做 one hot encoding

II. 建構模型

1. 首先定義模型中會使用到的 function，包含計算 mcrmse、attention、multi_head_attention、adj_attn 等
2. 建構一個 base model

   下列為粗略的網路架構：

   inputs → conv1ds → aggregation of neighborhoods → multi head attention → aggregation of neighborhoods → multi head attention → conv1d → predict
3. 使用所有的資料，包含 train、test 來建構 denoising auto encoder 的 pretrain model

Layer (type)	Output Shape	Param #	Connected to
node (InputLayer)	[(None, None, 39)]	0	
spatial_dropout1d (SpatialDropo	(None, None, 39)	0	node[0][0]
adj (InputLayer)	[(None, None, None,	0	
model (Functional)	(None, None, 135)	1906022	spatial_dropout1d[0][0] adj[0][0]
conv1d_112 (Conv1D)	(None, None, 64)	25984	model[0][0]
layer_normalization_90 (LayerNo	(None, None, 64)	128	conv1d_112[0][0]
dropout_88 (Dropout)	(None, None, 64)	0	layer_normalization_90[0][0]
leaky_re_lu_88 (LeakyReLU)	(None, None, 64)	0	dropout_88[0][0]
conv1d_113 (Conv1D)	(None, None, 64)	12352	leaky_re_lu_88[0][0]
layer_normalization_91 (LayerNo	(None, None, 64)	128	conv1d_113[0][0]
leaky_re_lu_89 (LeakyReLU)	(None, None, 64)	0	layer_normalization_91[0][0]
dropout_89 (Dropout)	(None, None, 64)	0	leaky_re_lu_89[0][0]
add_46 (Add)	(None, None, 64)	0	leaky_re_lu_88[0][0] dropout_89[0][0]
dense_3 (Dense)	(None, None, 39)	2535	add_46[0][0]
tf.math.subtract_1 (TFOpLambda)	(None, None, 39)	0	dense_3[0][0]
tf.__operators__.add (TFOpLambd	(None, None, 39)	0	dense_3[0][0]
tf.__operators__.add_1 (TFOpLam	(None, None, 39)	0	tf.math.subtract_1[0][0]
tf.math.multiply (TFOpLambda)	(None, None, 39)	0	node[0][0]
tf.math.log (TFOpLambda)	(None, None, 39)	0	tf.__operators__.add[0][0]
tf.math.subtract (TFOpLambda)	(None, None, 39)	0	node[0][0]
tf.math.log_1 (TFOpLambda)	(None, None, 39)	0	tf.__operators__.add_1[0][0]
tf.math.multiply_1 (TFOpLambda)	(None, None, 39)	0	tf.math.multiply[0][0] tf.math.log[0][0]
tf.math.multiply_2 (TFOpLambda)	(None, None, 39)	0	tf.math.subtract[0][0] tf.math.log_1[0][0]
tf.__operators__.add_2 (TFOpLam	(None, None, 39)	0	tf.math.multiply_1[0][0] tf.math.multiply_2[0][0]
tf.math.reduce_mean (TFOpLambda	()	0	tf.__operators__.add_2[0][0]
tf.math.negative (TFOpLambda)	()	0	tf.math.reduce_mean[0][0]

Total params: 1,947,149
Trainable params: 1,947,149
Non-trainable params: 0

4. 使用第 3 步的預訓練模型，再訓練出最後拿來預測結果的模型

Layer (type)	Output Shape	Param #	Connected to
node (InputLayer)	[(None, None, 39)]	0	
adj (InputLayer)	[(None, None, None,	0	
model_3 (Functional)	(None, None, 135)	1906022	node[0][0] adj[0][0]
conv1d_338 (Conv1D)	(None, None, 128)	51968	model_3[0][0]
layer_normalization_272 (LayerN	(None, None, 128)	256	conv1d_338[0][0]
dropout_266 (Dropout)	(None, None, 128)	0	layer_normalization_272[0][0]
leaky_re_lu_266 (LeakyReLU)	(None, None, 128)	0	dropout_266[0][0]
conv1d_339 (Conv1D)	(None, None, 128)	49280	leaky_re_lu_266[0][0]
layer_normalization_273 (LayerN	(None, None, 128)	256	conv1d_339[0][0]
leaky_re_lu_267 (LeakyReLU)	(None, None, 128)	0	layer_normalization_273[0][0]
dropout_267 (Dropout)	(None, None, 128)	0	leaky_re_lu_267[0][0]
add_139 (Add)	(None, None, 128)	0	leaky_re_lu_266[0][0] dropout_267[0][0]
dense_10 (Dense)	(None, None, 5)	645	add_139[0][0]

Total params: 2,008,427
Trainable params: 2,008,427
Non-trainable params: 0

5. 預測結果

Submission and Description	Private Score	Public Score
submission2.csv	0.36077	0.24860

六、結果分析與比較

I. 自我評估指標

為確認模型所預測出的數值有作用，因此前面我們自行產出一樣評估指標，以常態分布為底，mean 和 variance 則代入 training data 中，實際分解力和溶解率等的平均值和變異數。

	Mean	Variance
reactivity	0.3749	0.119
Deg_Mg_pH10	0.4463	0.12
Deg_Mg_50C	0.407	0.144

放入 kaggle 做驗證，右圖得到 Public Score 和 Private Sccore，可以確定之後模型預測出來的數值需優於此指標。

II. 特徵工程對模型的影響

	LGBMRegressor	CatBoostRegressor	NN
Public Score	0.42610	0.42610	0.42187
Private Score	0.48761	0.49204	0.49052

特徵數量:6

特徵工程

	LGBMRegressor	CatBoostRegressor	NN
Public Score	0.40375	0.41161	0.40526
Private Score	0.47775	0.61182	0.47592

特徵數量:25

上圖的 score 是來自 kaggle 上的評分結果。進行特徵工程之後，模型的表現皆有小幅度的改善，但成效並不是很明顯。 CatBoost 模型的 Public Score 有下降，Private Score 卻明顯上升，代表中間發生了過度擬和的問題，導致兩者的數據相差甚大。

III. 自訂序列相關特徵

對於 mRNA 構造而言，鹼基的排列方式是個重要的要素，因此我們嘗試新增自己定義的序列相關特徵，觀察是否影響模型表現。

我們嘗試在訓練 LGBM、CatBoost、NN 三種模型的時候，新增序列相關特徵「前後 10 個鹼基的資訊」，意思是每一筆資料為單個鹼基，但是我們會在每一筆資料內新增排序在該鹼基前後十個鹼基的資訊，來強調鹼基之間的排序關係。

	LGBMRegressor	CatBoostRegressor	NN
Public Score	0.40375	0.41161	0.40526
Private Score	0.47775	0.61182	0.47592

特徵數量:25

新增序列特徵

	LGBMRegressor	CatBoostRegressor	NN
Public Score	0.30354	0.31494	0.32934
Private Score	0.42279	0.61556	0.43778

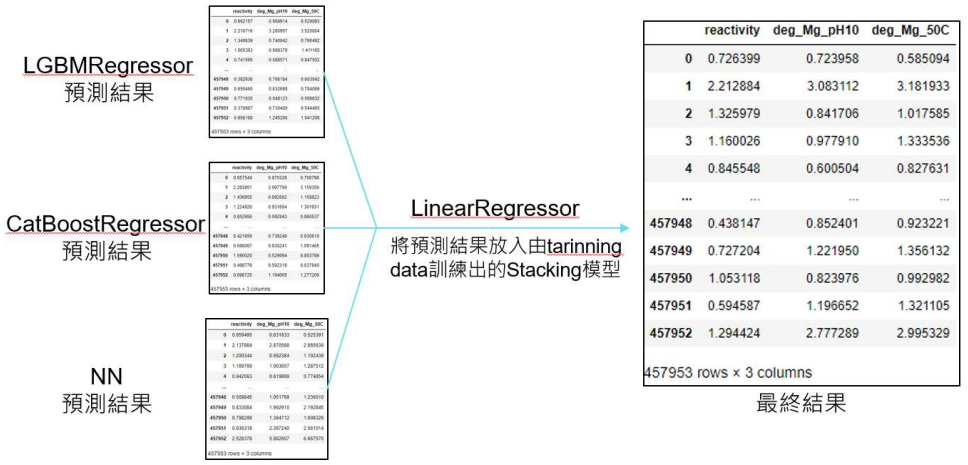
特徵數量:85

加入序列特徵之後，模型的 Public Score 有大幅度的改善。在特徵工程中我們主要是針對鹼基本身的特性去做處理，模型只有小幅度的改善，因此可以看出鹼基的序列方式才是影響降解率的關鍵。

IV. Ensemble Learning(集成學習)

針對 LGBM、CatBoost、NN 三種模型進行集成學習

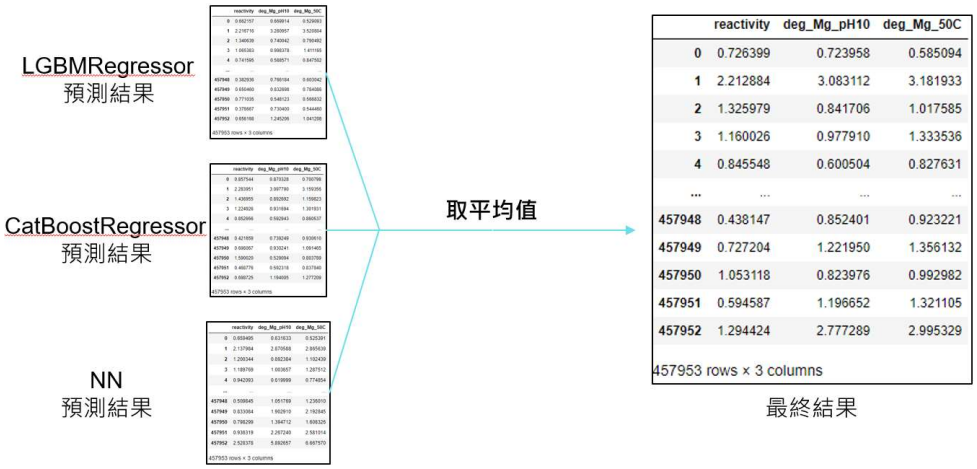
1. Stacking(堆疊法)



Submission and Description	Private Score	Public Score
to_csv_out_36.csv	0.73024	0.32908

這邊使用了雙層的堆疊模型，將 LGBM、CatBoost、NN 這三種模型的預測結果做為新的特徵，使用 LinearRegressor 模型重新訓練，得出新的預測結果。 可看出 Public Score 沒有太大的變化，反而 Private Score 大幅度上升，堆疊造成了嚴重的過度學習，使模型的泛化能力下降。

2. Bagging(引導聚集算法)



Submission and Description	Private Score	Public Score
mean.csv	0.45427	0.29847

這邊使用簡單的 Bagging 手法，取相同權重然後進行取平均值的動作，嘗試得到更好的結果。

進行 Bagging 之後，Public Score 的部分和 LGBM、CatBoost、NN 單個模型時相比，得到創新低的 0.29847，但是其實這意義不

大，因為 Public Score 是只用 9%的測試數據計算得出的分數，因此 Private Score 才是判斷模型成效的關鍵，而這邊未能看出 Private Score 有所改善。

V. 最佳預測成果

所有模型當中 RNN 表現為最好，明顯勝過其他的模型。 RNN 是一種使用結構遞歸的方式構建的神經網路，運用在 mRNA 序列資料上面能夠有效捕捉到鹼基序列的資訊，這樣的資訊很難光靠特徵工程的方式去取得。

Submission and Description	Private Score	Public Score
submission2.csv	0.36077	0.24860

Private Score 大概是排在 660/1636

Public Score 則是排在 791/1636

七、結論

I. 對預測結果進行比對後發現，特徵工程對模型的影響甚小，但是在加入自我定義的序列特徵之後對預測結果有大幅度改善，這點可以得出鹼基序列資訊對於預測降解率的重要性非常大。但是光靠人為定義的方式還是有所極限，運用 RNN 的方式讓模型自己去學習序列的特徵能夠有更好的表現。

II. 針對 LGBM、CatBoost、NN 三種模型使用 Ensemble Learning(集成學習)的時候成效並不是很好，嘗試增加模型數量、模型種類以及取不同權重，或許對預測結果可以有所改善。