Université de Paris

Master Bio-informatique
Parcours Biologie Informatique

Compétition Kaggle : Vaccin ARNm pour le COVID-19

Apprentissage, intelligence artificielle et optimisation (AIAO)

Auteur : Saad en-naimani

10/06/2021

Sommaire

I. Introduction

La maladie à coronavirus 2019 (COVID-19) est causée par un nouveau coronavirus connu sous le nom de coronavirus 2 du syndrome respiratoire aigu sévère (SRAS-CoV-2), qui est associé à plusieurs cas mortels dans le monde. La propagation rapide de ce pathogène et le nombre croissant de cas mettent en évidence le développement urgent de vaccins. Ce besoin urgent de vaccins contre la maladie en raison de la pandémie de SRAS-CoV-2 en cours a permis d'avoir plusieurs vaccins qui sont encore en étude. Parmi toutes les approches, un vaccin à base d'ARN messager (ARNm) est apparu comme une plateforme rapide et polyvalente pour répondre rapidement à ce défi. Ces vaccins à ARNm ont pris les devants en tant que candidats vaccins les plus rapides pour le COVID-19, mais actuellement, ils font face à des limitations potentielles clés. L'université de Stanford a découvert que l'un des plus grands défis à l'heure actuelle est de savoir comment concevoir des molécules d'ARN messager (ARNm) super stables. Les vaccins conventionnels (comme vos vaccins contre la grippe saisonnière) sont emballés dans des seringues jetables et expédiés sous réfrigération dans le monde entier, mais ce n'est actuellement pas possible pour les vaccins à ARNm. En effet, les chercheurs ont observé que les molécules d'ARN ont tendance à se dégrader spontanément. Il s'agit d'une limitation sérieuse - une seule coupure peut rendre le vaccin à ARNm inutile. Actuellement, on sait peu de choses sur les détails de l'endroit où dans le squelette d'un ARN donné est le plus susceptible d'être affecté. Sans cette connaissance, les vaccins à ARNm actuels contre le COVID-19 doivent être préparés et expédiés sous une réfrigération intense, et il est peu probable qu'ils atteignent plus d'une infime fraction d'êtres humains sur la planète à moins qu'ils ne puissent être stabilisés.

L'objectif est donc de créer un modèle qui permet de prédire le taux de dégradation de chaque base de chaque séquence ARNm d'un sous ensemble issues d'un jeu de données contenant 3029 séquences d'ARN. Nous disposons de trois informations qualitatives et structurales décrivant les séquences d'ARNm : la séquence, l'appariement des bases et la prédiction de boucles. Nous allons donc créer trois modèles de prédictions en ne prenant que la séquence comme variable explicative dans un premier temps, puis un deuxième modèle qui prendra en compte la séquence et les informations d'appariements de bases et enfin un troisième modèle qui prendra en compte les trois variables explicatives. Puis nous comparerons ces trois modèles afin de voir quel est le meilleur et surtout si l'introduction d'informations structurales améliore la prédiction.

II. Matériels et méthodes

A. Jeu de données

Les scientifiques de Stanford ont des données sur 3029 séquences d'ARN de longueur 107.
Pour des raisons techniques, les mesures ne peuvent pas être effectuées sur les bases finales de ces séquences d'ARN, nous avons donc des données expérimentales (ground truth) dans 5 conditions « reactivity », « deg_Mg_ph10 », « deg_Mg_50c », « deg_ph10 », « deg_50c » pour

les 68 premières bases. 629 ont été répartis de ces 3029 séquences pour un test public afin de permettre une évaluation continue tout au long de la compétition, dans le classement public. Ces séquences, dans test.json, ont en outre été filtrées sur la base de trois critères « sequence », « structure » et « predicted loop » pour garantir que ce sous-ensemble n'est pas dominé par un grand cluster de molécules d'ARN avec des données médiocres, ce qui pourrait biaiser le classement public. Les 2400 séquences restantes pour lesquelles nous avons des données sont dans train.json. Pour notre score final et le plus important (Private Leaderboard), les scientifiques de Stanford effectuent des mesures sur 3005 nouveaux ARN, qui ont des longueurs un peu plus longues de 130 bases. Pour ces données, nous nous attendons à avoir des mesures pour les 91 premières bases, encore une fois manquant les extrémités de l'ARN. Ces séquences constituent encore 3005 des 3634 séquences de test.json.

B. Pipeline

Avant de créer notre modèle, nous avons nettoyé et traiter nos données afin d'avoir un aperçu de la distribution des valeurs de dégradation. Pour commencer nous avons supprimé toutes les séquences qui avaient un rapport signal/noise inférieur à 1. En effet un rapport inférieur à 1 signifie qu'il y a eu des erreurs au niveau expérimental, et ces échantillons peuvent conduire à une mauvaise prédiction. Ensuite nous avons regardé la redondance de nos séquences grâce au web server CD_hit. En effet, si le taux d'identité entre les différentes séquences est trop élevé et s'il y a trop de séquences similaires notre modèle va mal apprendre et donc mal prédire. En sortie, nous obtenons un fichier qui découpe notre jeu en plusieurs clusters. Un cluster signifie que les séquences sont considérées comme similaires et un taux d'identité y est associé. Cette redondance pourra être traitée en pondérant les séquences avec « sample_weight » sur keras.

C. Méthodes de construction de modèles utilisées

Différents types de réseau neuronal existent, pour notre prédiction, nous avons utilisé un réseau neuronal de type GRU (Gated Recurrent Unit). GRU est une variante des réseaux de type RNN. RNN est le type de réseau qui convient le plus à notre type de variable à prédire, cependant ce réseau à des limitations. En effet un RNN utilise la méthode de la descente du gradient afin de mettre à jour les poids entre ses neurones. Cependant au fur et à mesure de l'apprentissage, la mise à jour du pois ne fait plus très bien et par conséquent le modèle apprend mal et peut oublier des données anciennes. La mémoire d'un RNN est donc plutôt courte. Pour cela, nous avons utilisé GRU, qui a une meilleure architecture et qui permet de résoudre les problèmes de mémoire à court et long termes. Pour le loss, nous avons implémenté la fonction MCRMSE (mean columnwise root mean square error). Le MCRMSE est une moyenne de toutes les valeurs RMSE pour chaque colonne du jeu de données. Et le RMSE, est une moyenne de la différence entre les valeurs observées et les valeurs prédites. Ensuite nous avons joué sur plusieurs paramètres tels que la fonction d'optimisation, d'activation, le nombre de couches et. En effet, pour les fonctions d'optimisations, nous avons testé différents algorithmes tels que stochastic gradient descent (SGD) et Adam. Pour les fonctions d'activations nous avons essayé avec les fonctions LeakyRelu et Sigmoid.

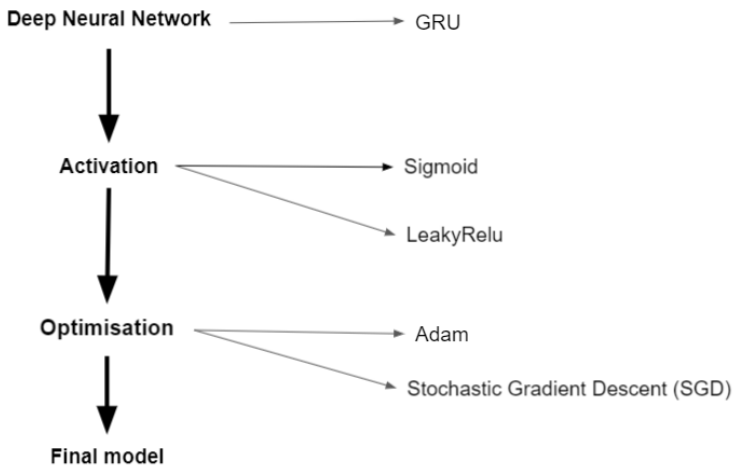

Figure 1: schéma des méthodes utilisées.

D. Analyse et conception

1. Nettoyage et préparation des données

Notre nettoyage des données consiste à enlever toutes séquences dont le rapport signal/noise est inférieur à 1. Après suppression de ces séquences, nous obtenons un jeu de 2097 échantillons. Ensuite, nous avons utilisé le web server CD_hit pour retrouver si nos séquences sont redondantes.

La préparation des données consiste à encoder nos variables qualitatives « sequence », « structure » et « predicted_loop » en utilisant l'encodage one-hot pour obtenir le résultat ci-dessous. Notre jeu de données est maintenant contient 22 colonnes.

One_hot seq	One hot strucure	encoding predicted loop
[[[1, 0, 0, 0]], [[1, 0, 0, 0]], [[0, 1, 0, 0]...	[[[1, 0, 0]], [[1, 0, 0]], [[1, 0, 0]], [[1, 0...	[[[1, 0, 0, 0, 0, 0, 0]], [[1, 0, 0, 0, 0, 0, ...
[[[1, 0, 0, 0]], [[1, 0, 0, 0]], [[0, 1, 0, 0]...	[[[1, 0, 0]], [[1, 0, 0]], [[1, 0, 0]], [[1, 0...	[[[1, 0, 0, 0, 0, 0, 0]], [[1, 0, 0, 0, 0, 0, ...

Figure 2: scéen des colonnes encodé du jeu train

2. Analyse exploratoire

L'analyse exploratoire consiste à étudier les 3 variables qualitatives par rapport au 5 conditions « reactivity », « deg_Mg_ph10 », « deg_Mg_50c », « deg_ph10 », « deg_50c ».

a) Représentation des données de séquences

On a cherché la valeur maximale des 5 conditions pour chaque séquences d'ARNm, après on l'associé avec une base, puis on a compté le nombre de chaque bases (**Figure 4**)

	base	number
0	A	856
3	C	36
1	G	1010
2	U	195
	base	number
1	A	613
2	C	83
0	G	1201
3	U	200

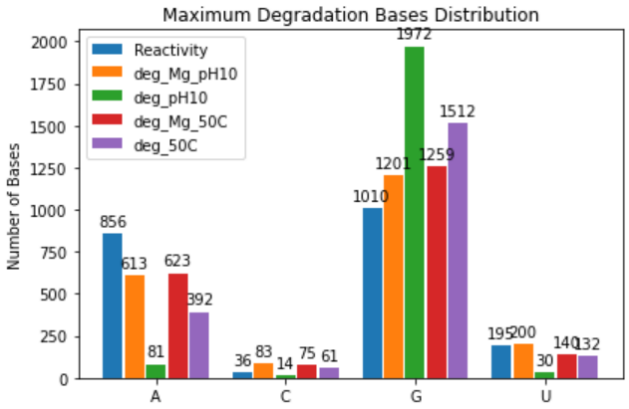

Figure 4 : tableau (réactivité et deg_Mg_pH10) et histogramme du nombre de bases en fonction de la valeur maximale de dégradation du jeu train.

On voit que la base G est la base qui a le taux de dégradation le plus élevé, donc on peut déduire que la base G est la plus susceptible à la dégradation pour toutes les variables de dégradations suivi de la base A.

b) Représentation des données structurales

On a cherché la valeur maximale des 5 conditions pour chaque séquences d'ARNm, après on l'associé avec une structure, puis on a compté le nombre de chaque structure (**Figure 5**)

	structure	number
1	(	94
2	)	39
0	.	1964

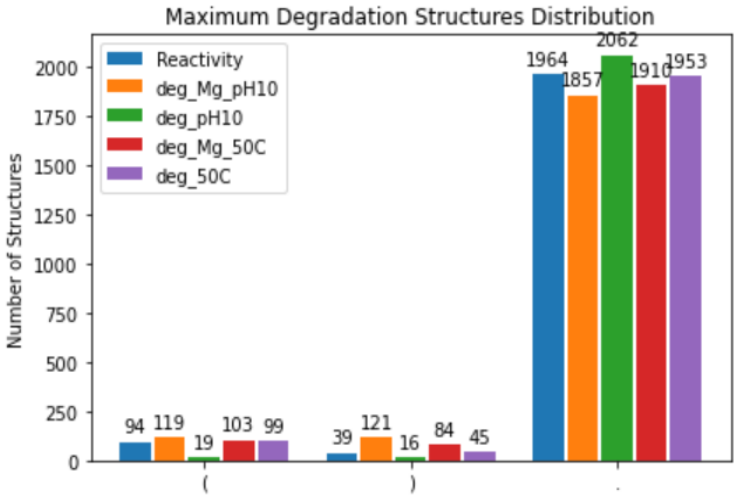

Figure 5: tableau(réactivité) et histogramme du nombre de structures en fonction de la valeur maximale de dégradation du jeu train.

D'après le plot de la distribution des structures on voit que les bases non appariées sont les plus susceptibles à la dégradation.

c) Représentation des données de prédiction des boucles

On a cherché la valeur maximale des 5 conditions pour chaque séquences d'ARNm, après on l'associé avec un type de boucles, puis on a compté le nombre de chaque type de boucles (**Figure 6).**

	loop_type	number
3	B	11
0	E	1742
1	H	110
4	I	48
2	M	18
6	S	144
5	X	24

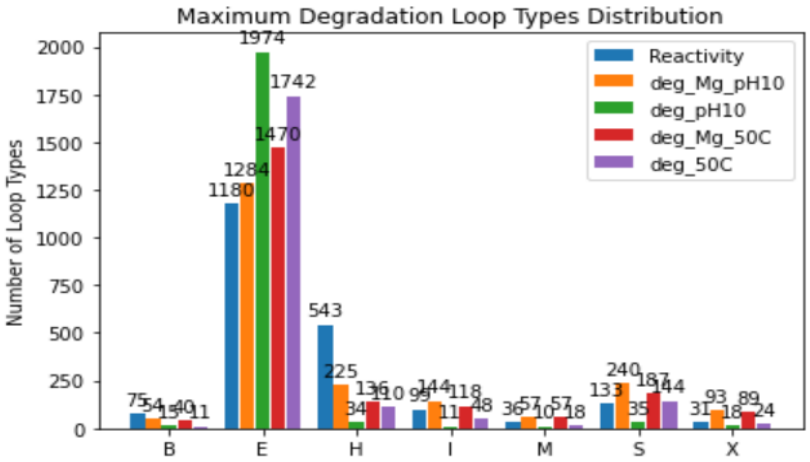

Figure 6: tableau (réactivité) et histogramme du nombre de types de boucle en fonction de la valeur maximale de dégradation du jeu train.

Pour le plot de la distribution des types de boucles, on peut voir que les boucles E sont les plus susceptibles à la dégradation.

3. Paramètre utilisées

Après avoir choisis les méthodes d'activation, Nous avons testé pour chaque fonction d'activation, un optimiseur et différents nombres de couches. Nous avons essayé avec 1, 2 puis 3 couches avec un nombre d'epoch de 45, en choisissant un batch size de 64.

III. Résultat

A. Les meilleurs modèles de prédiction

(Le reste des résultats voir annexe)

1. Une variable explicative (Modèle 1)

Paramètres	Loss	Val_loss
Adam LeakyRelu 3 couche Batch 64 epoch 45	0.2081	0.2388

Tableau 1 : Résultat du modèle 1 prédit à partir des informations de séquence

2. Deux variables explicatives (Modèle 2)

Paramètres	Loss	Val_loss
Adam LeakyRelu 2 couche Batch 64 epoch 45	0.2084	0.2407

Tableau 2 : résultat du modèle 2 prédit à partir des informations de séquence et de structure

3. Trois variables explicatives (Modèle 3)

Paramètres	Loss	Val_loss
Adam LeakyRelu 3 couche Batch 64 epoch 45	0.2089	0.2382

Tableau 3 : résultat du modèle 3 prédit à partir des informations de séquence, de structure et de prédiction des boucles

D'après les tableaux ci-dessus, on remarque donc qu'en ajoutant des informations supplémentaires (structure et/ou prédiction de boucles) la valeur du loss et du val_loss augmente, et dans tous les cas la valeur du loss reste inférieur à la valeur du val_loss.

IV. Conclusion

Pour conclure après avoir testé différents modèles (**Annexe**) nous concluant que le meilleur modèle de prédiction que nous avons obtenu est le modèle 1. En effet il ne semble pas présenter de surapprentissage (voire un léger surapprentissage) et à des valeurs de loss et val_loss basses mais pas excellente.

Annexe

Modèle	loss	val_loss
Adam Sigmoïde 1 variable « séquence » Batch 64 Une couche epochs 45	**0.3092**	**0.3152**
Adam LeakyRelu 1 variable « séquence » Batch 64 Une couche epochs 45	**0.2211**	**0.2509**
SGD Sigmoïde 1 variable « séquence » Batch 64 Une couche epochs	**0.4758**	**0.4595**
SGD LeakyRelu 1 variable « séquence » Batch 64 Une couche epochs	**0.4419**	**0.4272**
Adam Sigmoïde 2 variable « seq+structure » Batch 64 Une couche epochs	**0.3092**	**0.3152**
Adam LeakyRelu 2 variable « seq+structure » Batch 64 Une couche epochs	**0.2211**	**0.2509**
SGD Sigmoïde 2 variable « seq+structure »	**0.4758**	**0.4595**

Batch 64 Une couche epochs		
SGD LeakyRelu 2 variable « seq+structure » Batch 64 Une couche epochs	**0.4410**	**0.4272**
Adam Sigmoïde 3 variable « seq+structure+boucle » Batch 64 Une couche epochs	**0.3092**	**0.3152**
Adam LeakyRelu 3 variable « seq+structure+boucle » Batch 64 Une couche epochs	**0.4419**	**0.4272**
SGD sigmoid 3 variable « seq+structure+boucle » Batch 64 Une couche epochs	**0.4767**	**0.4595**
SGD LeakyRelu 3 variable « seq+structure+boucle » Batch 64 Une couche epochs	**0.4406**	**0.4266**
Adam sigmoid 1 variable « séquence » Batch 64 2 couche epochs	**0.2999**	**0.3079**
Adam LeakyRelu 1 variable « séquence » Batch 64	**0.2089**	**0.2409**

2 couche epochs		
SGD Sigmoïde 1 variable « séquence » Batch 64 2 couche epochs	**0.4787**	**0.4622**
SGD LeakyRelu 1 variable « séquence » Batch 64 2 couche epochs	**0.4438**	**0.4280**
Adam Sigmoïde 2 variable « seq+structure » Batch 64 2 couche epochs	**0.2999**	**0.3079**
Adam LeakyRelu 2 variable « seq+structure » Batch 64 2 couche epochs	**0.2084**	**0.2407**
SGD Sigmoïde 2 variable « seq+structure » Batch 64 2 couche epochs	**0.4787**	**0.4622**
SGD LeakyRelu 2 variable « seq+structure » Batch 64 2 couche epochs	**0.4443**	**0.4280**
Adam Sigmoïde 3 variable «seq+str+boucle» Batch 64 2 couche	**0.2999**	**0.3079**

epochs		
Adam **LeakyRelu** **3 variable** **«seq+str+boucle»** **Batch 64** **2 couche** **epochs**	**0.2089**	**0.2408**
SGD **sigmoid** **3 variable** **«seq+str+boucle»** **Batch 64** **2 couche** **epochs**	**0.4787**	**0.4622**
SGD **LeakyRelu** **3 variable** **«seq+str+boucle»** **Batch 64** **2 couche** **epochs**	**0.4438**	**0.4280**
Adam **Sigmoïde** **1 variable « séquence »** **Batch 64** **3 couche** **epochs**	**0.3033**	**0.3074**
Adam **LeakyRelu** **1 variable « séquence »** **batch 64** **3 couche** **epochs**	**0.2081**	**0.2388**
SGD **sigmoid** **1 variable « séquence »** **Batch 64** **3 couche** **epochs**	**0.4816**	**0.4639**
SGD **LeakyRelu** **1 variable « séquence »** **Batch 64** **3 couche** **epochs**	**0.4471**	**0.4298**
Adam	**0.3033**	**0.3074**

Sigmoïde **2 variable « seq+structure »** **Batch 64** **3 couche** **epochs**		
Adam **LeakyRelu** **2 variable « seq+structure »** **Batch 64** **3 couche** **epochs**	**0.4473**	**0.4298**
SGD **Sigmoïde** **2 variable « seq+structure »** **Batch 64** **3 couche** **epochs**	**0.4816**	**0.4639**
SGD **LeakyRelu** **2 variable « seq+structure »** **Batch 64** **3 couche** **epochs**	0.4473	0.4298
Adam **sigmoid** **3 variable « seq+structure+boucle »** **Batch 64** **3 couche** **epochs**	0.3041	0.3074
Adam **LeakyRelu** **3 variable « seq+structure+boucle »** **Batch 64** **3 couche** **epochs**	0.2089	0.2382
SGD **Sigmoïde** **3 variable « seq+structure+boucle»** **Batch 64** **3 couche** **epochs**	**0.4816**	**0.4639**

SGD **LeakyRelu** **3 variable «** **seq+structure+boucle »** **Batch 64** **3 couche** **epochs**	0.4473	0.4298

Références

https://www.kaggle.com/c/stanford-covid-vaccine
https://www.kaggle.com/onodera/covid-ae-pretrain-gnn-attn-cnn
https://www.kaggle.com/isaienkov/openvaccine-eda-feature-engineering-modeling