Tison Maxime
Tep Florian

Université de Paris - M2 BI
Apprentissage, IA et Optimisation

Prédiction d'une Réponse Biologique

(Projet Kaggle)

Code accessible à partir du lien GitHub : https://github.com/flotep/bioresponse

I - Introduction

L'objectif de ce projet est de développer le modèle le plus optimal possible pour permettre de prédire une réponse biologique à partir d'informations sur les propriétés physico-chimiques de molécules. Dans les données disponibles pour réaliser ces prédictions, chacune des molécules étudiées est définie par des descripteurs sous forme de variables quantitatives comprises entre 0 et 1. Ces descripteurs représentent les propriétés des molécules qui ont été calculées pour capturer certaines de leurs caractéristiques comme leur taille, leur forme ou encore leur constitution. Chacun des descripteurs est identifié par un numéro, leur nature précise n'a pas été renseignée.

Dans les données fournies pour entraîner notre modèle, chaque molécule est associée à une « activité » observée expérimentalement. Cette valeur d'activité indique si oui (activité = 1) ou non (activité = 0) l'utilisation de la molécule a induit la réponse biologique recherchée.

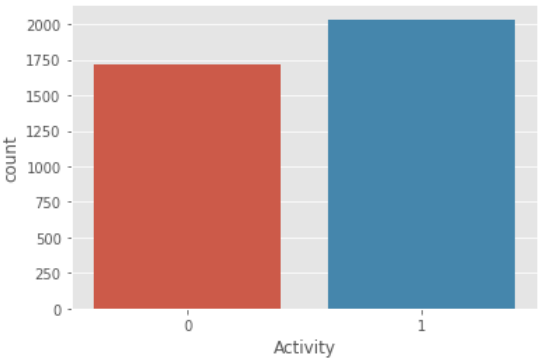
Afin de réaliser les prédictions attendues, notre modèle devra effectuer une régression à partir des valeurs des descripteurs pour calculer pour chaque molécule sa probabilité d'induire une réponse biologique. Plus la valeur prédite sera proche de 1, plus on s'attendra à ce que la molécule étudiée induise une réponse biologique.

Les probabilités prédites par le modèle seront évaluées suivant la métrique de la perte logistique (logistic loss), avec le meilleur score obtenu = 0.37355

II - Analyse exploratoire des données (EDA)

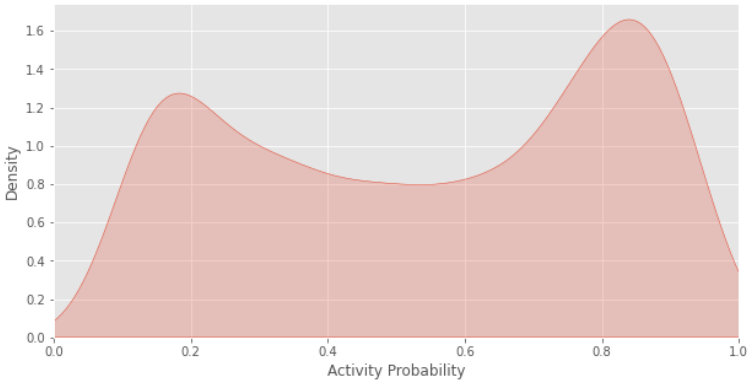
Les données mises à disposition pour le projet sont des tableaux au format CSV, l'un d'eux est dédié à l'entraînement du modèle « train », et le second au test du modèle « test ». Au sein de ces tableaux, chaque ligne correspond à une molécule et chaque colonne correspond à un descripteur de ces molécules.

Le jeu de données pour la phase de « train » contient 3751 molécules différentes contre 2501 molécules pour les données de « test ». Chacun des tableaux présente 1776 descripteurs, la 1ère colonne des données de « train » contenant les valeurs d'« activité ». Sur les 3751 molécules, 2 034 ont une



activité égale à 1, soit 54%, et 1 777 ont une activité égale à 0. La distribution peut donc être considérée comme équivalente.

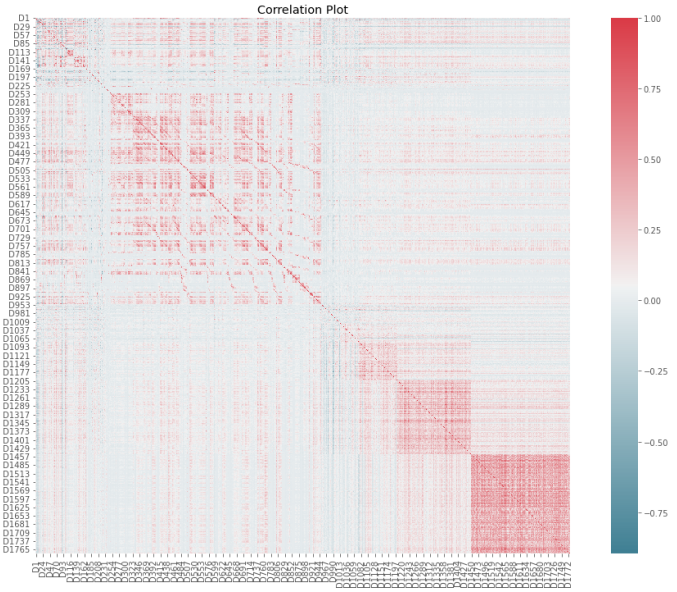
Les prédictions attendues pour les données de test sont répertoriées dans un 3eme tableau, la distribution des probabilités est la suivante : . Les probabilités d'activités de ces molécules sont concentrées autour de 0.2 et 0.8, avec également une densité importante entre 0.4 et 0.6.



Aucune valeur manquante n'a été trouvée, chacune des molécules présentes dans les données est décrite par 1776 descripteurs différents. Parmi ces descripteurs, 942 sont de types float compris entre 0 et 1 (soit 53%), et 834 sont de type entier avec des valeurs égales à 0 ou 1.

```
Valeurs manquantes 0
Colonnes de type float Index(['D1', 'D2', 'D3', 'D4', 'D5', 'D6', 'D7', 'D8', 'D9', 'D10',
       ...
       'D942', 'D943', 'D944', 'D945', 'D946', 'D947', 'D948', 'D949', 'D950',
       'D951'],
      dtype='object', length=942)
Colonnes de type int Index(['D23', 'D27', 'D28', 'D51', 'D72', 'D94', 'D170', 'D568', 'D858',
       'D952',
       ...
       'D1767', 'D1768', 'D1769', 'D1770', 'D1771', 'D1772', 'D1773', 'D1774',
       'D1775', 'D1776'],
      dtype='object', length=834)
```

Une analyse bivariée est réalisée avec une matrice de corrélation entre tous les descripteurs selon la méthode de Pearson. Nous pouvons distinguer 4 groupes de variables différents qui apparaissent être corrélées positivement entre elles. Nous pouvons remarquer également quelques groupes plus réduit de variables corrélées négativement. Toutes ces corrélations suggèrent que certains descripteurs se rapportent à des caractéristiques de molécules liées entre elles ou de même type sur le plan biologique.



Afin de résoudre le problème de régression de ce projet, notre choix de modèle de prédictions s'est porté vers divers types d'approches : les arbres de décisions (Extra-trees et Random Forest), l'algorithme des K plus proches voisins (KNN), les machines à vecteurs de support (SVM), ainsi que les réseaux de neurones artificiels (Fully Connected Neural Network).

La problématique majeure pour réaliser nos prédictions à partir de ces données est que le nombre de variables pour chacune des molécules est très important. Dans un premier temps, nous avons cherché à standardiser leurs distributions pour les rendre comparables entre elles et faciliter l'apprentissage de nos modèles. Pour cela, nous avons effectué un pré-traitement de nos données avec une mise à l'échelle entre 0 et 1 suivant la méthode de Min-Max Scaling.

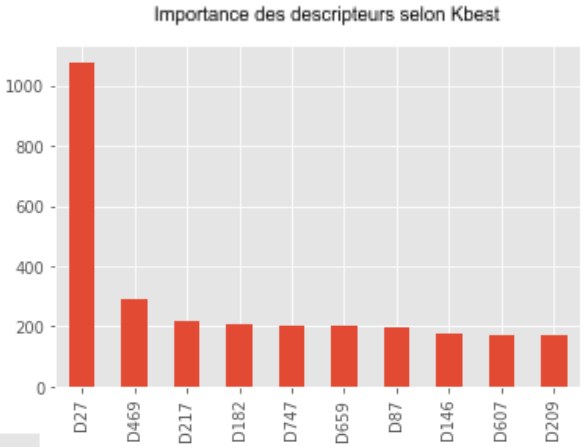
Suite à cela, pour optimiser les performances, simplifier les modèles que l'on veut construire, et dans le but d'éviter la redondance des informations dans nos données, nous avons cherché à sélectionner les descripteurs les plus pertinents (Feature Selection) ou bien à réduire la dimensionnalité de l'espace des données (Dimensionality reduction).

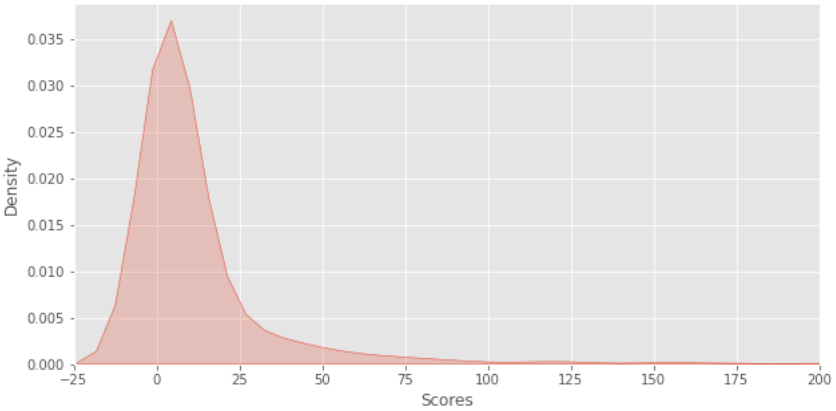
III - Sélection de Features

Deux méthodes ont été étudiées pour effectuer la sélection des descripteurs dans le jeu de données.

a) Sélection des top k variables (SelectKBest)

Cette première méthode de sélection permet de filtrer les descripteurs en fonction de leur importance. Cette importance est définie en fonction d'un score qui a été attribué à chaque variable en calculant leur coefficient de corrélation de Pearson. A partir de la distribution des scores pour les 1776 descripteurs, nous avons choisi de sélectionner ceux dont le score était supérieur à 11.2 (valeur du 3ème quartile), soit environ 443 descripteurs.






```
count    1776.000
mean       12.945
std        35.440
min         0.000
25%         0.972
50%         4.319
75%        11.189
max      1076.620
Name: Score, dtype: object
```

```
Train data shape avant sélection des features : (3751, 1776)
Train data shape après sélection des kbest features : (3751, 443)
```

b) Sélection des variables selon leur variance (Variance Threshold)

On remarque avec la méthode des tops k variables que l'importance des descripteurs semblent très équivalente entre eux, à l'exception de D27. Cette méthode de sélection n'est donc peut être pas judicieuse, c'est pourquoi nous avons choisi d'étudier une seconde méthode de feature selection. Celle-ci permet de filtrer les descripteurs en fonction de leur variance, en estimant qu'une variable avec une variance quasi-nulle aura une influence négligeable sur l'apprentissage d'un modèle. Un seuil de variance de 0.01 a été choisi en rapport avec le support de nos variables qui est compris entre 0 et 1. Avec un tel seuil, on ne conserve qu'environ 964 descripteurs, ce qui indique qu'un très grand nombre des descripteurs de départs avaient une variance très faible.

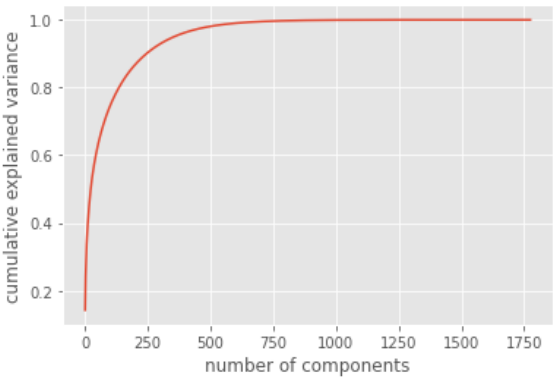
```
Train data shape avant Variance Threshold: (3751, 1776)
Train data shape après Variance Threshold (seuil = 0.01): (3751, 964)
```

IV - Réduction de la dimensionnalité

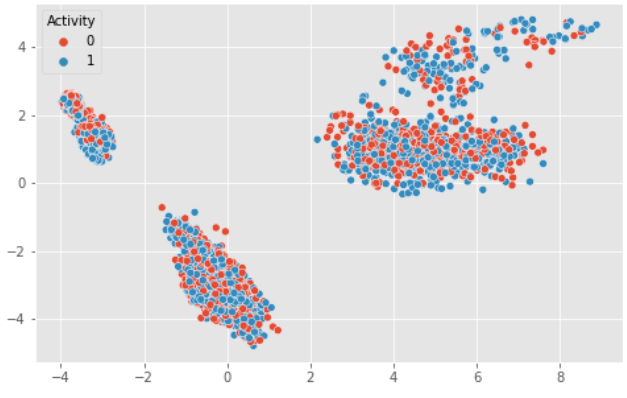
Dans un second temps, nous avons étudié s'il était possible de réduire l'espace à haute dimension de nos données à l'aide de trois méthodes différentes.

a) Analyse en composante principale (ACP)

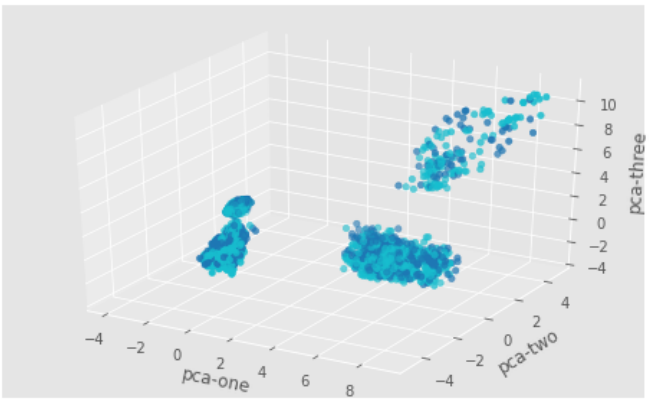
Tout d'abord, nous avons utilisé l'algorithme de transformation linéaire ACP dans l'objectif de projeter les variables originales de nos données dans un sous-espace en retenant le plus d'informations possible. Pour cela, l'algorithme tente de trouver les composants principaux qui maximisent la variance dans de nouveaux sous-espaces.



En analysant la variance cumulée expliquée par les composants principaux obtenus (graphe), on remarque qu'environ 350 composants sont nécessaires pour expliquer plus de 95% de la variance des données.



Représentation 2D des composantes 1 et 2 de l'ACP



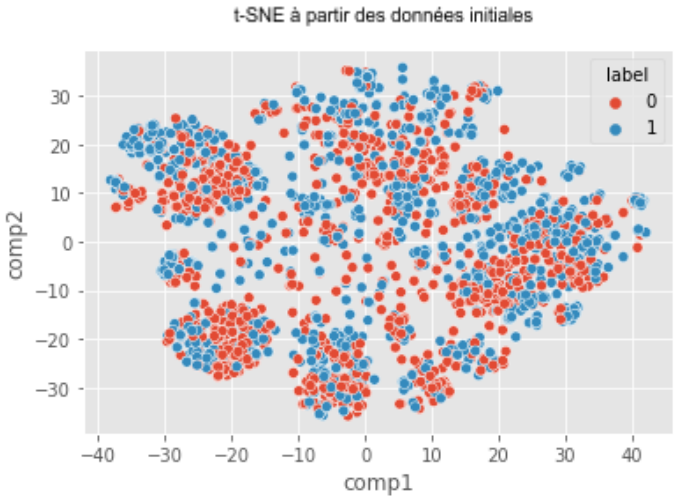
Représentation 3D des composantes 1,2 et 3 de l'ACP

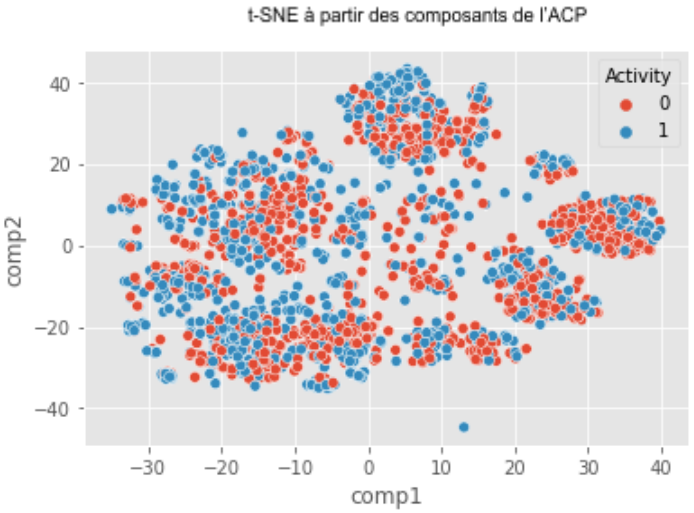
Représentation 2D selon les 5 principales composantes de l'ACP

La représentation graphique des données en 2D ou 3D dans les composants qui expliquent le mieux la variance ne révèle pas de distinction claire entre les molécules qui induisent une réponse biologique et celles qui n'en induisent pas. La projection de nos données dans ce sous-espace ne permet pas d'identifier des classes de molécules.

b) t-Distributed Stochastic Neighbor Embedding (t-SNE)

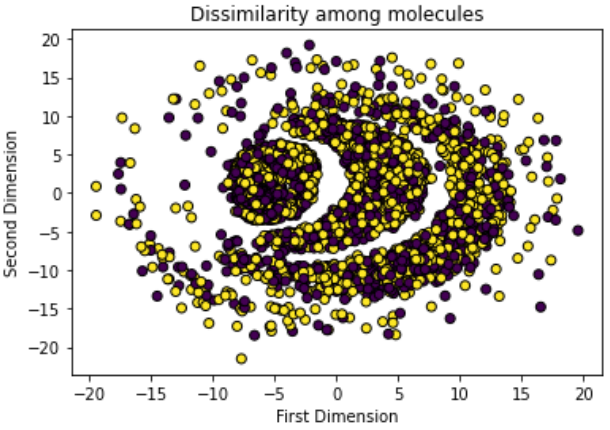
t-SNE est une méthode non-linéaire probabiliste de réduction de dimensionnalité. A partir des distances euclidiennes entre deux points, l'algorithme calcul des probabilités conditionnelles. Un test de Student est ensuite appliqué sur ces probabilités pour calculer la similarité entre chacun des points. Nous avons réalisé un t-SNE à deux composants sur les données dans l'espace d'origine ou dans l'espace réduit par l'ACP. En ajustant le paramètre de perplexité de l'algorithme (ici égale à 100), aucune ségrégation claire des classes de molécules n'a pu être identifiée dans les deux espaces.





c) Mise à l'échelle multidimensionnelle (MDS)

La méthode non linéaire de réduction de dimensionnalité MDS permet de visualiser le niveau de similarité des individus dans notre jeu de données. L'objectif est d'identifier les coordonnées cartésiennes des individus à partir de leur matrice de distances, de telle sorte à ce que la distance (ou dissimilarité) entre chaque paire d'individus soit préservée autant que possible. L'analyse du MDS à deux dimensions que nous avons réalisée ne permet également pas d'observer une séparation des classes des molécules, les groupements d'individus visibles révèlent des amalgames de molécules qui induisent une activité, avec des molécules qui n'en induisent pas.



La réduction de la dimensionnalité réalisée avec ces trois premières méthodes ne permet donc pas d'aboutir à une représentation des données d'origines dans un sous-espace où des classes de molécules peuvent être distinguées aisément. On suppose donc que l'utilisation de ces données aux dimensions réduites aura une faible influence sur l'apprentissage des modèles de Machine Learning.

d) Auto-Encodeur

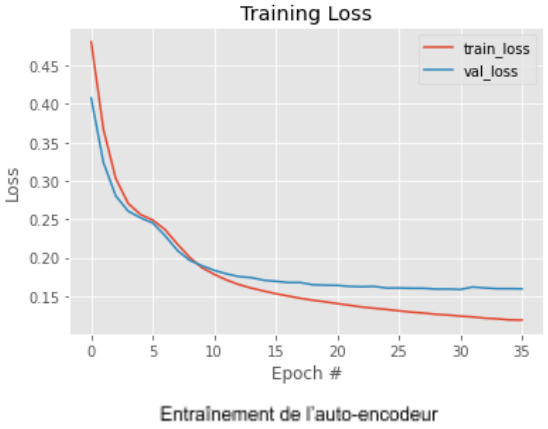
Toujours dans l'objectif de représenter nos données dans un sous-espace aux dimensions réduites, une dernière méthode alternative basée sur l'encodage des données a été testée. Pour cela, nous avons utilisé l'architecture basique d'un auto-encodeur avec une couche d'entrée, une couche cachée de 2 neurones, et une couche de sortie de la même dimension que la couche d'entrée.

Layer (type)	Output Shape	Param #
input_7 (InputLayer)	[(None, 1776)]	0
dense_24 (Dense)	(None, 2)	3554
dense_25 (Dense)	(None, 1776)	5328

Total params: 8,882
Trainable params: 8,882
Non-trainable params: 0

Les réseaux neuronaux de ce type sont souvent utilisés pour construire une nouvelle représentation d'un jeu de données, comme des images par exemple. Leur architecture est constituée de deux parties : l'encodeur et le décodeur. L'encodeur à pour rôle de traiter les données afin de construire une nouvelle représentation de celles-ci "encodées", alors que le décodeur reçoit ces représentations et les traitent afin d'essayer de reconstruire les données initiales.

Ici, nous nous intéressons à la représentation "encodées" des données, nous récupérons donc les informations de sortie de l'encodeur, à savoir la représentation des données par rapport aux deux neurones de la couche cachée de l'auto-encodeur, une fois celui-ci entraîné avec les données initiales.



Entraînement de l'auto-encodeur

Représentation des données prédite par l'encodeur

Une fois encore, nous ne remarquons pas de ségrégation claire des données (voir scatter plot ci-dessus), néanmoins, une distinction est visible entre les molécules induisant une activité, situées majoritairement dans la 1ère moitié de la droite, des molécules qui n'induisent pas d'activité, situées plutôt dans la seconde partie de la droite. La représentation "encodées" des données à partir des prédictions de l'encodeur semble donc être plus adaptée pour réduire la dimensionnalité de notre jeu de données.

V - Prédictions à partir de modèles de Machine Learning

a) Méthodes d'ensembles d'arbres

Ces méthodes d'ensembles sont décomposées en un large nombre d'arbres de décisions, chacun des arbres ayant une influence sur la décision finale du modèle. Chaque arbre de décisions part d'un nœud racine, les nœuds internes représentent les features, les branches représentent les règles de décisions, et les décisions (ici la prédiction d'une probabilité) sont représentées par les feuilles. Dans le cas de notre problème de régression, la moyenne arithmétique des prédictions de chaque arbre permet d'obtenir la valeur de la prédiction finale.

❖ <u>Extra Trees :</u>

L'algorithme Extra Trees fonctionne en construisant un grand nombre d'arbres de décisions à partir de l'ensemble du jeu de données réservé à l'apprentissage. A chaque point de séparation rencontré lors de la construction d'un arbre, les features sont échantillonnées aléatoirement et le point de séparation du nœud est choisi au hasard. Les hyper paramètres retenus pour notre régression par Extra Trees sont : 400 arbres dans l'ensemble, chaque arbre à une taille maximale de 20 nœuds, et au moins 1 échantillon est requis pour créer un nouveau point de séparation dans un nœud. Après entraînement avec les données initiales (1776 features), le score de cross validation obtenu en suivant la métrique de la perte logistique = **0.51**.

- Le score du modèle entraîné avec les features filtrées par leur variance (950 features): **0.50**.
- Le score du modèle entraîné avec les k top features sélectionnées (450 features): **0.46**.

- Le score du modèle entraîné avec les composantes principales de l'ACP (350 composantes): **0.53**.
- Le score du modèle entraîné avec la représentation encodée des données issue de l'auto-encodeur (2 dimensions) : **0.48**.

Données initiales (1776 features)	Variance Threshold (964 features)	Kbest features (443 features)	ACP (350 composantes)	Données encodées (2 dimensions)
0.513	0.499	**0.462**	0.532	0.478

❖ Random Forest :

Cette méthode d'ensemble développe chaque arbre de décision à partir d'un échantillon bootstrap issu de l'ensemble des données d'apprentissage, obtenu par rééchantillonnage avec remise d'un échantillon initial. Lors de la construction des arbres, les features sont échantillonnées aléatoirement à chaque point de séparation comme pour l'algorithme Extra Trees. Cependant, le point de séparation retenu est le point optimal et non un point aléatoire. Avec les mêmes hyperparamètres que ceux utilisés pour la méthode précédente, les scores de perte logistique obtenus par chacun des modèles sont les suivants :

Données initiales (1776 features)	Variance Threshold (964 features)	Kbest features (443 features)	ACP (350 composantes)	Données encodées (2 dimensions)
0.488	**0.455**	0.462	0.543	0.468

b) Méthode des k plus proches voisins (k-NN)

La régression par la méthode k-NN permet de prédire le résultat (ici la probabilité d'une molécule à induire une activité) d'une entrée en calculant la distance entre cette donnée d'entrée et toutes les données d'apprentissage du modèle. Ainsi, la valeur prédite correspondra à la moyenne des résultats obtenus par les données des plus proches voisins de cette entrée. Les hyper paramètres optimaux pour ce modèle k-NN ont été estimés grâce à la méthode de Grid Search avec cross validation : le nombre de voisins les plus proches considérés est de 7, le point de chaque voisin étant 'uniforme'.

En ajustant le modèle sur les données contenant toutes les features d'origines (1776 descripteurs), nous obtenons un score de perte logistique = **2.04**. Ce score se dégrade considérablement par rapport aux méthodes d'ensembles d'arbres car ce type de modèle est très peu performant sur les données à hautes dimensions. En ajustant le modèle k-NN sur des données lorsque les features ont été sélectionnées ou lorsque leur dimensionnalité a été réduite, on observe que les performances des prédictions s'améliorent mais restent bien loin de celles des méthodes d'ensembles d'arbres.

Données initiales (1776 features)	Variance Threshold (964 features)	Kbest features (443 features)	ACP (350 composantes)	Données encodées (2 dimensions)
2.04	**1.04**	1.27	1.40	1.31

c) Méthode de Machine à Vecteurs de Support (SVM)

Un algorithme de type SVM à pour objectif de séparer les données en classes en créant une frontière (hyperplane) entre ces classes de manière à ce que la distance (marge) entre les groupes de données soit maximale. Pour séparer linéairement les données, le SVM utilise des méthodes permettant de projeter les données dans un espace de dimension supérieure. Dans le cadre de notre problème de régression, nous avons utilisé un modèle SVR (Support Vector Regression) qui cherche lui à trouver la frontière contenant le maximum de données possible. Cette frontière sera ainsi utilisée pour prédire nos valeurs de sorties (la probabilité d'une molécule à induire une réponse biologique).

Les hyper paramètres optimaux retenus pour notre modèle SVR sont les suivants : kernel de type 'rbf', kernel coefficient de type 'scale' (gamma), paramètre de régularisation égal à 1 (C), une marge autour de la frontière de 0.2 (epsilon). Les scores de pertes logistiques obtenus avec les différents modèles ont montré que celui entraîné par les données encodées était significativement plus performant. Ces hyperparamètres ont donc été ajustés afin d'optimiser le score obtenu : C = 60, epsilon = 0.15.

Les résultats des différents modèles SVR sont résumés dans le tableau suivant :

Données initiales (1776 features)	Variance Threshold (964 features)	Kbest features (443 features)	ACP (350 composantes)	Données encodées (2 dimensions)
0.582	0.570	0.458	0.597	**0.416**

d) Les Réseaux de Neurones Artificiels

Notre problème de régression peut finalement être adressé par un modèle basé sur des réseaux de neurones. La phase d'entraînement d'un réseau de neurones permet à chaque observation d'ajuster les poids des connexions au sein du réseau de manière à réduire l'erreur de prédiction du modèle. On va ainsi itérer plusieurs fois sur l'intégralité du jeu de données (epoch) jusqu'à ce que les résultats de l'algorithme convergent, signe que le réseau ne peut plus apprendre davantage. L'un des enjeux d'un tel modèle est d'optimiser sa vitesse d'apprentissage de manière à qu'elle soit rapide quand les poids sont mal réglées et lente quand on approche du bon réglage.

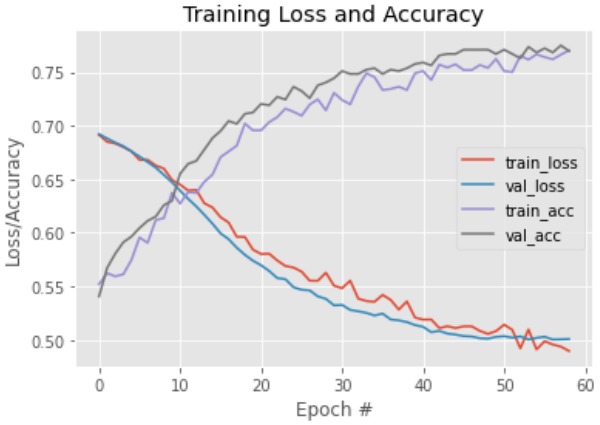
Un réseau de neurones fully connected (FCNN) a été choisi pour réaliser nos prédictions. Au sein de ce type de modèle, tous les neurones d'une couche sont connectés à chacun des neurones de la couche suivante, aussi appelé couche dense.

L'organisation du réseau que nous avons utilisé est le suivant : une couche pour les données d'inputs, trois couches cachées composées respectivement de 64, 32 et 32 neurones (activation relu) chacune précédée d'une couche de dropout, et une couche de sortie composée d'un unique neurone (activation sigmoid) chargée de renvoyer le résultat de la prédiction.

Les résultats obtenus lors du réglage des paramètres du réseau ont révélé que l'augmentation du nombre de paramètres du modèle (ajout de couches cachées, neurones supplémentaires,...) aboutit à la dégradation du score des prédictions. La fonction de perte utilisée au cours de l'apprentissage est la Binary Cross-Entropy, la fonction d'optimisation est la descente de gradient stochastique (SGD) avec un learning rate de 0.01, la taille d'un batch est de 64, le nombre d'epoch est limité à 100, une fonction de rappel 'Early Stopping' est utilisée pour arrêter l'apprentissage lorsque la 'validation loss' n'évolue plus (patience de 5).

L'architecture de notre modèle a été entraînée avec différentes dimensions de notre jeu de données afin de comparer les résultats obtenus. L'évaluation des prédictions de notre modèle sur les données de tests dans chacun des cas est le suivant :

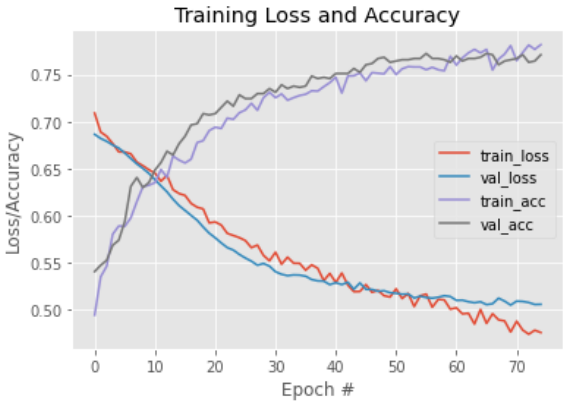
- Modèle ajusté sur les données dans l'espace initial : perte logistique = **0.55**



Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 1776)]	0
dropout (Dropout)	(None, 1776)	0
dense (Dense)	(None, 64)	113728
dropout_1 (Dropout)	(None, 64)	0
dense_1 (Dense)	(None, 32)	2080
dropout_2 (Dropout)	(None, 32)	0
dense_2 (Dense)	(None, 32)	1056
dense_3 (Dense)	(None, 1)	33

Total params: 116,897
Trainable params: 116,897
Non-trainable params: 0

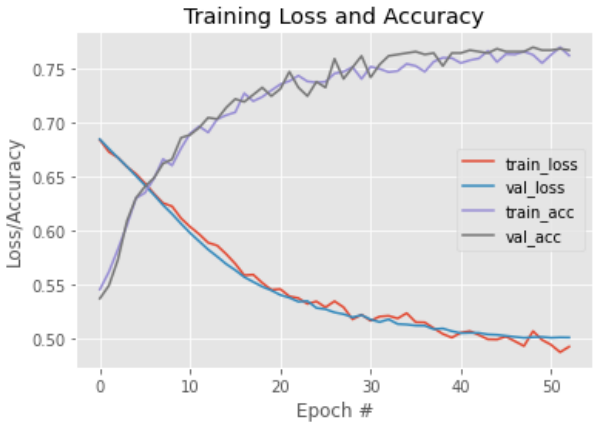
- Modèle ajusté sur les données réduites par le seuil de variance : perte logistique = **0.55**



Layer (type)	Output Shape	Param #
input_3 (InputLayer)	[(None, 964)]	0
dropout_6 (Dropout)	(None, 964)	0
dense_8 (Dense)	(None, 64)	61760
dropout_7 (Dropout)	(None, 64)	0
dense_9 (Dense)	(None, 32)	2080
dropout_8 (Dropout)	(None, 32)	0
dense_10 (Dense)	(None, 32)	1056
dense_11 (Dense)	(None, 1)	33

Total params: 64,929
Trainable params: 64,929
Non-trainable params: 0

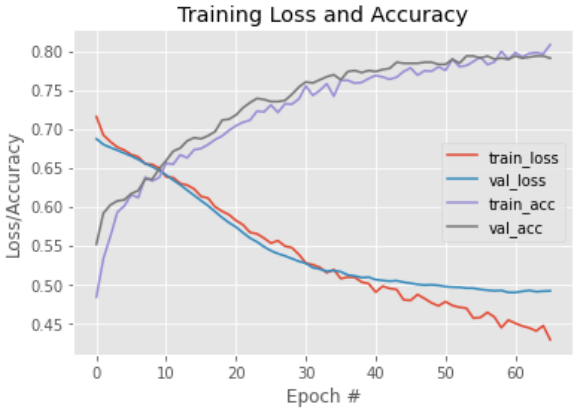
❖ Modèle ajusté sur les données réduites par les k top variables : perte logistique = **0.54**



Layer (type)	Output Shape	Param #
input_5 (InputLayer)	[(None, 443)]	0
dropout_12 (Dropout)	(None, 443)	0
dense_16 (Dense)	(None, 64)	28416
dropout_13 (Dropout)	(None, 64)	0
dense_17 (Dense)	(None, 32)	2080
dropout_14 (Dropout)	(None, 32)	0
dense_18 (Dense)	(None, 32)	1056
dense_19 (Dense)	(None, 1)	33

Total params: 31,585
Trainable params: 31,585
Non-trainable params: 0

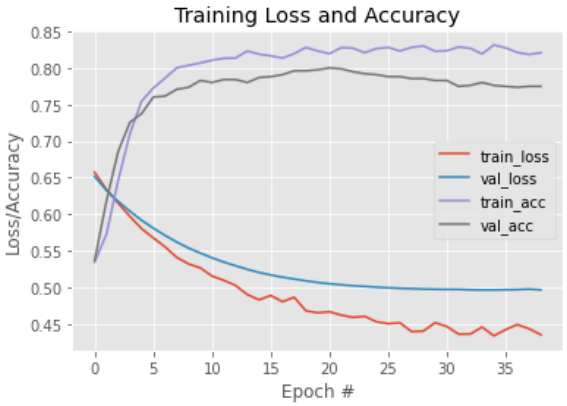
❖ Modèle ajusté sur les données dans le sous-espace de l'ACP : perte logistique = **0.57**



Layer (type)	Output Shape	Param #
input_6 (InputLayer)	[(None, 350)]	0
dropout_15 (Dropout)	(None, 350)	0
dense_20 (Dense)	(None, 64)	22464
dropout_16 (Dropout)	(None, 64)	0
dense_21 (Dense)	(None, 32)	2080
dropout_17 (Dropout)	(None, 32)	0
dense_22 (Dense)	(None, 32)	1056
dense_23 (Dense)	(None, 1)	33

Total params: 25,633
Trainable params: 25,633
Non-trainable params: 0

❖ Modèle ajusté sur la représentation encodée des données à partir d'un auto-encodeur : perte logistique **= 0.57**



Layer (type)	Output Shape	Param #
input_4 (InputLayer)	[(None, 2)]	0
dropout_3 (Dropout)	(None, 2)	0
dense_8 (Dense)	(None, 64)	192
dropout_4 (Dropout)	(None, 64)	0
dense_9 (Dense)	(None, 32)	2080
dropout_5 (Dropout)	(None, 32)	0
dense_10 (Dense)	(None, 32)	1056
dense_11 (Dense)	(None, 1)	33

Total params: 3,361
Trainable params: 3,361
Non-trainable params: 0

Le résumé des scores obtenus avec les prédictions des différents réseaux de neurones sur le jeu de données test est le suivant :

Données initiales (1776 features)	Variance Threshold (964 features)	Kbest features (443 features)	ACP (350 composantes)	Données encodées (2 dimensions)
0.556	0.551	**0.549**	0.574	0.572

VI - Conclusion

La plupart des méthodes de Machine Learning étudiées dans le cadre de ce projet ont révélées des performances relativement similaires, la moins adaptée d'entre elles étant le modèle des k plus proches voisins. Concernant les autres méthodes, les scores de pertes logistiques obtenus avec des hyper paramètres optimaux semblent tous converger vers une valeur entre 0.45 et 0.60, et cela quelque soit les features sélectionnées ou les dimensions des données. Ceci pourrait être expliqué par le fait que les features ont toutes une part d'importance relativement similaire, et que la représentation des données dans un espace réduit ne permet pas dans la plupart des cas d'optimiser l'apprentissage des modèles car la ségrégation de classes n'est pas distincte.

Néanmoins, parmi les méthodes étudiées, le meilleur score a été obtenu en utilisant un modèle de régression SVM entraîné à partir de la représentation encodée des données en 2 dimensions (0.416). Cette observation laisse suggérer que l'hyperplane optimal pour réaliser nos prédictions est généré par le SVR lorsque la dimensionnalité des données est considérablement réduite.

Nous pouvons également noter les performances des méthodes d'ensembles d'arbres Extra Trees et Random Forest qui semblent, tout comme le SVM, particulièrement adaptées pour prédire la probabilité d'une molécule à induire une réponse biologique à partir du jeu de données dont nous disposons.

Cependant, d'autres travaux sur ce projet ont montré qu'il était possible d'obtenir des modèles encore plus performants, permettant de converger vers des scores inférieurs à 0.40. Ces modèles utilisent d'autres méthodes de Machine Learning telles que XGBoost, CatBoost, ou encore LightGBM qui sont des implémentations optimisées de l'algorithme d'arbres de renforcement du gradient (gradient boosting). Ces techniques d'apprentissage se sont révélées particulièrement performantes sur les données complexes et volumineuses telles que celles étudiées lors de ce travail.