Université de Genève

Data Mining
13X011

TP : Projet final

Author: Edin Sulejmani

E-mail: Edin.Sulejmani@etu.unige.ch

Author: Joao Filipe Costa da Quinta

E-mail: Joao.Costa@etu.unige.ch

June 25, 2021

UNIVERSITÉ
DE GENÈVE
FACULTÉ DES SCIENCES
Département d'informatique

Introduction

Dans ce projet nous avons reçu un grand set de données, concernant des molécules chimiques et des propriétés de ces mêmes molécules. Le but était de développer des modèles qui seraient entraînées et testées à l'aide du set de données.

Représentation des données

Quand on a reçu le data set, on a aussi reçu un utils.py, avec des fonctions qui changeaient la représentation de la molécule elle même, car il y a des molécules avec plus ou moins d'éléments, mais pour implémenter un modèle, on a besoin que toute donnée ait la même taille.
Dans le data set, les molécules sont représentées à l'aide du protocole SMILES. Voici deux molécules quelconques, molécule (a) C, et la molécule (b) CCC1=CC=CO1, on ne peut pas se servir des cette notation pour nos modèles. Du coupon transforme SMILES en une autre notation:

La première notation est une 'simple' traduction de lettre en chiffre, mais qui donne une valeur à l'absence de valeur, comme ça toute molécule du data set a la même taille:

```
C ->  [5, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13]
C#CC1=CC=CO1 ->  [5, 2, 5, 5, 6, 3, 5, 5, 3, 5, 7, 6, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13]
```

La deuxième notation est différente, et pas vraiment pratique, car la donnée est très grande, ce qui rendera notre modèle moins performant, et elle e nous donne pas vraiment plus d'information:

```
C :
[[0 0 0 0 0 1 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 1]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 1]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 1]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 1]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 1]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 1]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 1]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 1]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 1]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 1]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 1]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 1]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 1]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 1]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 1]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 1]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 1]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 1]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 1]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 1]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 1]]
```

```
C#CC1=CC=CO1 :
[[0 0 0 0 0 1 0 0 0 0 0 0 0 0]
 [0 0 1 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 1 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 1 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 1 0 0 0 0 0 0 0]
 [0 0 0 1 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 1 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 1 0 0 0 0 0 0 0 0]
 [0 0 0 1 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 1 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 1 0 0 0 0 0 0]
 [0 0 0 0 0 0 1 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 1]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 1]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 1]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 1]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 1]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 1]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 1]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 1]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 1]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 1]]
```

Par contre, depuis cette deuxième notation, on peut additionner les colonnes entre elles, et on se retrouvera avec une notation qui nous indique combien d'éléments de chaque type se trouvent dans la molécule:

```
C :
[ 0  0  0  0  0  1  0  0  0  0  0  0  0 21]
C#CC1=CC=CO1 :
[ 0  0  1  2  0  6  2  1  0  0  0  0  0 10]
```

Dans ce TP on va utiliser les représentations suivantes:

```
C ->  [5, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13]
C#CC1=CC=CO1 ->  [5, 2, 5, 5, 6, 3, 5, 5, 3, 5, 7, 6, 13, 13, 13, 13, 13, 13, 13, 13, 13]
```

```
C :
[ 0  0  0  0  0  1  0  0  0  0  0  0  0 21]
C#CC1=CC=CO1 :
[ 0  0  1  2  0  6  2  1  0  0  0  0  0 10]
```

Selon l'attribut qu'on cherche à trouver, on utilisera l'une ou l'autre, car dans la première notation, on garde l'ordre des éléments dans la molécule, ce qui peut être important, alors que dans l'autre notation, on est plus précis, on verra plus tard en quoi cela pourra nous aider.
Définissons la représentation ordonné (qui garde l'ordre des éléments dans la molécule), et la représentation quantitative (qui met en avant la quantité de chaque élément de la molécule).

Attributs

- logP: represents a measure of the tendency of a compound to move from the aqueous phase into lipids
- Number of rotatable bonds (RBN): the number of bonds which allow free rotation around themselves
- Molecular weight (MW): the weight of a molecule based on the atomic masses of all atoms in the molecule
- Number of the rings (RN): the number of connected sets of atoms and bonds in which every atom and bond is a member of a cycle

Dans le but de développer le meilleur modèle possible, il faudrait avoir des connaissances dans le domaine de la chimie, ce n'est pas notre cas.

L'un de ceux qu'on comprend, est le 3ème, le poids moléculaire, qui est défini par la somme du poids de chaque élément appartenant à la molécule, il est donc clair que pour cet attribut, il faudra utiliser la 2ème notation, qui nous indique combien d'éléments spécifiques la molécule contient, sans se soucier de leur position dans la molécule elle même.

Les autres attributs on va essayer le même modèle avec les deux représentations différentes, et voir laquelle donne les meilleurs résultats.

Baseline - Linear regression

Pour baseline on a choisit un modèle simple, performant et un minimum efficace.

Le modèle cherche à calculer un vecteur w, chaque élément de ce vecteur correspond à la valeur qui a le moins d'erreur pour toute donnée d'entraînement. Sans surprise le modèle qui prédit le poids était presque parfait, si on utilisait la bonne représentation de données, c'est du au fait que, mettre des poids à chaque x_i et c'est exactement ce qu'il faut faire pour avoir un résultat parfait. Pour les autres attributs il était tout de même un peu moins performant.
La représentation quantitative a été la meilleur pour tout attribut, mais les résultats ne sont pas impressionnants pour le RBN.

logP maxiter = 500, lr = invscaling, moyenne erreur = 0.343. Si on doit deviner la valeur 1.0, alors le modèle va donner 0.656, ou 1.34, en moyenne.

RBN maxiter = 500, lr = invscaling, error rate = 56%

MW maxiter = 1000, lr = constant, moyenne erreur = 0.017. Si on doit deviner la valeur 150.0, alors le modèle va donner 149.98, ou 150.013, en moyenne.

RN maxiter = 500, lr = invscaling, error rate = 5%

Pour les attributs discrets, plutôt que utiliser le classifier, on a eu des meilleurs résultats en faisant le regressor puis faisant l'arrondissement de résultats.

Ces arguments ont eu les meilleurs résultats parmi les combinaisons suivantes:

max_iter [100, 500, 1000, 2000, 5000]

learning_rate ['constant', 'optimal', 'invscaling', 'adaptive']

KNN - nearest neighboor

Ce modèle ne marche que sur les attributs de classe, ou discret, il est donc que utilisable pour RBN et RN. Il a un fonctionnement très simple, on a une formule qui calcule la distance ou taille de chaque donné, et puis quand on veut predict on calcule de la même façon la taille de notre valeur à predict, et on cherche les n voisins plus proches (des données d'entraînement) en distance, on regarde les classes de ces voisins et on attribue au x qu'on cherche à predict la classe la plus représenté parmi les voisins.

Chez ce modèle la représentation avec les meilleurs résultats pour l'attribut RBN était la représentation ordonnée, pour RN toujours la représentation quantitative.

RBN Représentation ordonnée - neighboors = 1, error rate = 54%.

RN Représentation quantitative - neighboors = 5, error rate = 7%

Pour obtenir ces résultats j'ai essayé des modèles pour les deux attributs avec les deux représentations, avec neigboors = [1,2,3,4,5].

Neural Networks

Le modèle neural network, comme le modèle de répression linéaire cherche à calculer un vecteur de poids, sauf que l'erreur à chaque étape se fait diffèrament, avec de la back propagation.

logP hiddenlayersizes=(20,6), lr = 0.001, moyenne erreur = 0.26. Si on doit deviner la valeur 1.0, alors le modèle va donner 0.74, ou 1.26, en moyenne.

RBN hiddenlayersizes=(15,4), lr = 0.001. error rate = 45%

MW hiddenlayersizes=(1,5), lr = 0.001, moyenne erreur = $2e-06$. On peut tout simplement dire qu'il est parfait.

RN hiddenlayersizes=(20,5), lr = 0.001. error rate = 35%

J'ai malheureusement fait une erreur, et fait le modèle pour RN, avec la représentation ordonnée, du coup je vais ignorer les résultats de la boucle, car c'est toujours l'autre représentation qui a eu les meilleurs résultats pour cet attribut.

Ces arguments ont eu les meilleurs résultats parmi les combinaisons suivantes:

learning_rate [0.1, 0.01, 0.001, 0.0001]

couches [1, 5, 10, 15, 20]

rones par couche [1, 2, 3, 4, 5, 6]

En essayent de faire des améliorations chez RBN et RN, j'ai relancé avec quelques arguments différents, et les résultats ont été les suivants : (cette fois ci avec la représentation quantitative pour RN).

RBN maxiter=2000, hiddenlayersizes=(200,15), lr = 0.001, alpha=1e-5. error rate = 38%

RN maxiter=2000, hiddenlayersizes=(200,15), lr = 0.001, alpha=1e-5. error rate = 5%

Ce qui est déjà bien meilleur.

Gradiant Boosting Regressor

Pour le GBRegressor, l'apprentissage se fait par optimisation d'une loss function (d'où le nom de 'gradiant'). Ici c'est une méthode de 'Tree' donc on peut choisir le nombre de noeuds futurs à explorer par la méthode pour une efficacité maximale. Le nombre par défaut est maxDepth=3. Le modèle de base est calculé avec les paramètres 'x' et 'y', puis l'input de chaque decision Tree sera l'output de l'erreur du Tree précédent et le paramètre 'x'. L'erreur est une prédiction nommée 'residual value' et donc calculé par chaque Tree.
Ici, nous avons décidé de itérer sur les différentes loss functions disponibles : ["ls", "lad", "huber", "quantile"] 'ls' se réfère au least squares regression, 'lad' (least absolute deviation) est une méthode hautement robuste basé sur l'ordre de l'information des inputs. 'huber' est une combinaison de 'ls' et 'lad'. 'quantile' est une regression quantile. Ainsi que sur le learning rate : [10, 1, 0.1, 0.01, 0.001, 0.0001, 0.00001]. Nous avons obtenu les résultats optimales suivants pour ces paramètres :

logP lossfunction = lad, lr = 1, moyenne erreur = 0.267. Si on doit deviner la valeur 1.0, alors le modèle va donner 0.73 ou 1.268, en moyenne.

RBN lossfunction = ls, lr = 1, error rate = 58.43%

MW lossfunction = ls, lr = 1, moyenne erreur = 0.66. Si on doit deviner la valeur 150.0, alors le modèle va donner 149.34 ou 150.66, en moyenne.

RN lossfunction = huber, lr = 1, error rate = 5.4%

Suite à des tests avec différentes maxDepth, on remarque que le learning rate n'a pas d'impact sur l'erreur, le learning rate optimal pour les 4 attributs est '1'. Cependant on voit qu'on a presque une loss function optimale différente pour chaque attribut. On va alors tester à nouveau avec plusieurs maxDepth, avec learning rate = 1. Les loss functions restent les mêmes, les différents maxDepth seront : [3, 4, 5, 7, 9].

Les nouveaux résultats obtenus sont :

logP lossfunction = lad, bestdepth = 3, moyenne erreur = 0.267. (aucun changement)

RBN lossfunction = ls, bestdepth = 4, error rate = 58.36% (gain de 0.07%)

MW lossfunction = huber, bestdepth = 7, moyenne erreur = 0.62. (best loss function = huber + gain de 0.04 en moyenne)

RN lossfunction = huber, bestdepth = 3, error rate = 5.4% (aucun changement)

On remarque des petits changement dans les loss functions ainsi que dans les erreurs. Un gain assez négligeable pour le temps de calcul de l'optimalité qui a prit environ 30 minutes. Cependant, en trouvant un maxDepth optimal, cette méthode peut potentiellement être très efficace.

Support Vector Regression

Pour une régression simple, on essaie de minimiser l'erreur rate, alors qu'avec SVR on va essayer de 'fit' une l'erreur dans un certain seuil. On utilise une fonction kernel pour mapper les données de petites dimensions sur des données de dimensions supérieurs. On utilise un Hyper plane qui est notre ligne de séparation entre les classes qui va nous permettre de prédire les valeurs target/continues. On a aussi des Boundary line qui seront notre 'marge' autour de l'Hyper plane, c'est elles qui vont séparer nos classes. Enfin on a les support vectors

qui sont les points des données les plus proches des Bourndary line.

Ici, le temps d'exécution d'une seule itération était d'environ 3 heures. C'est le dernier modèle que nous avons testé et donc par manque de temps nous avons choisi de ne pas faire de tests extrêmes pour chaque paramètres. Le but sera de voir l'efficacité de cette méthode pour une itération et même si nous n'auront forcément pas les paramètres optimales, décider si cette méthode à un bon rapport temps/efficacité. Si c'est le cas, il serait intéressant de chercher à optimiser la méthode, sinon il sera préférable de ne pas utiliser cette méthode pour ce modèle.

Les paramètres utilisés sont les suivants : kernel : 'rbf', C : '1.0', epsilon : '0.2'. Voici les résultats non-optimal, pour une itération :

logP kernel = 'rbf', C = 1, epsilon = 0.2 moyenne erreur = 0.267. Si on doit deviner la valeur 1.0, alors le modèle va donner 0.73 ou 1.268, en moyenne.

RBN kernel = 'rbf', C = 1, epsilon = 0.2, error rate = 57.47%

MW kernel = 'rbf', C = 1, epsilon = 0.2, moyenne erreur = 0.9. Si on doit deviner la valeur 150.0, alors le modèle va donner 149.1 ou 150.9, en moyenne.

RN kernel = 'rbf', C = 1, epsilon = 0.2, error rate = 7.29%

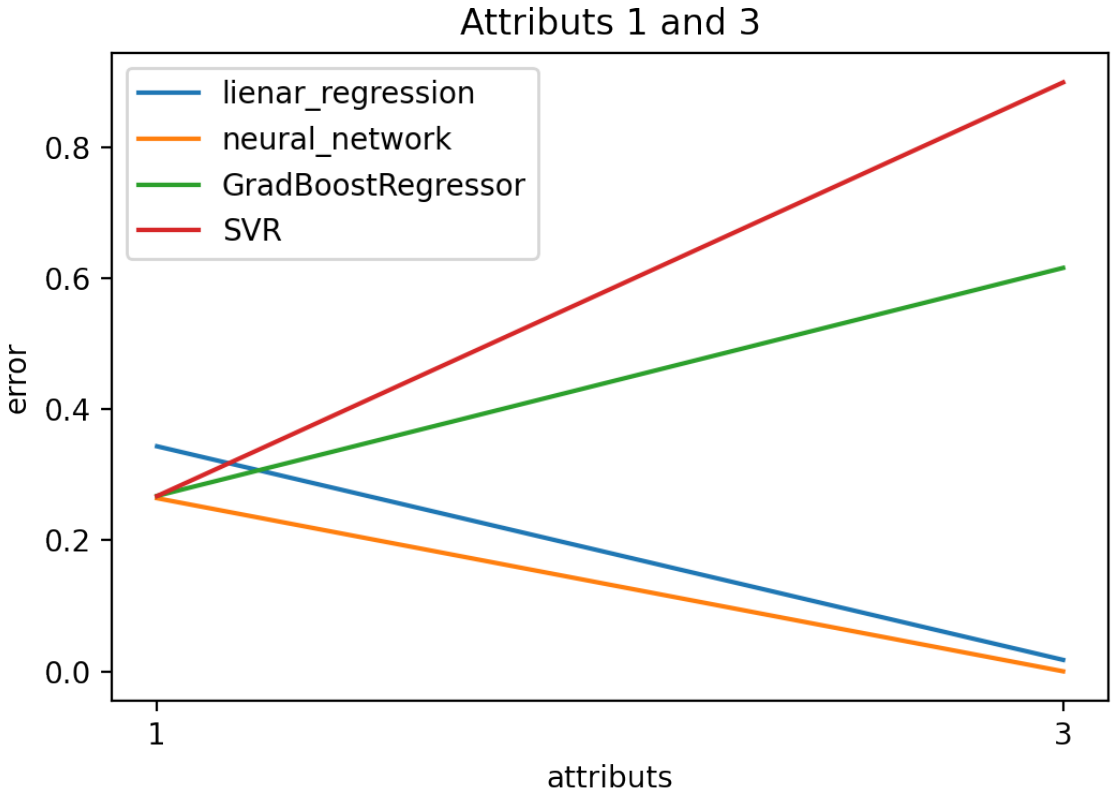
On remarque que le SVR avec les paramètres testés n'est pas vraiment si efficace que ça. Pour le temps de calcul nécessaire et l'efficacité en comparaison avec les autres méthodes utilisé, le SVR n'est meilleur pour aucun des attributs. Dans une situation où l'optimale est recherchée, il serait pratique de voir si cette méthode peut faire mieux. Dans le cas contraire, l'inefficacité en temps et en erreur est un point énorme négatif à la méthode et n'est pas recommandé pour notre modèle.

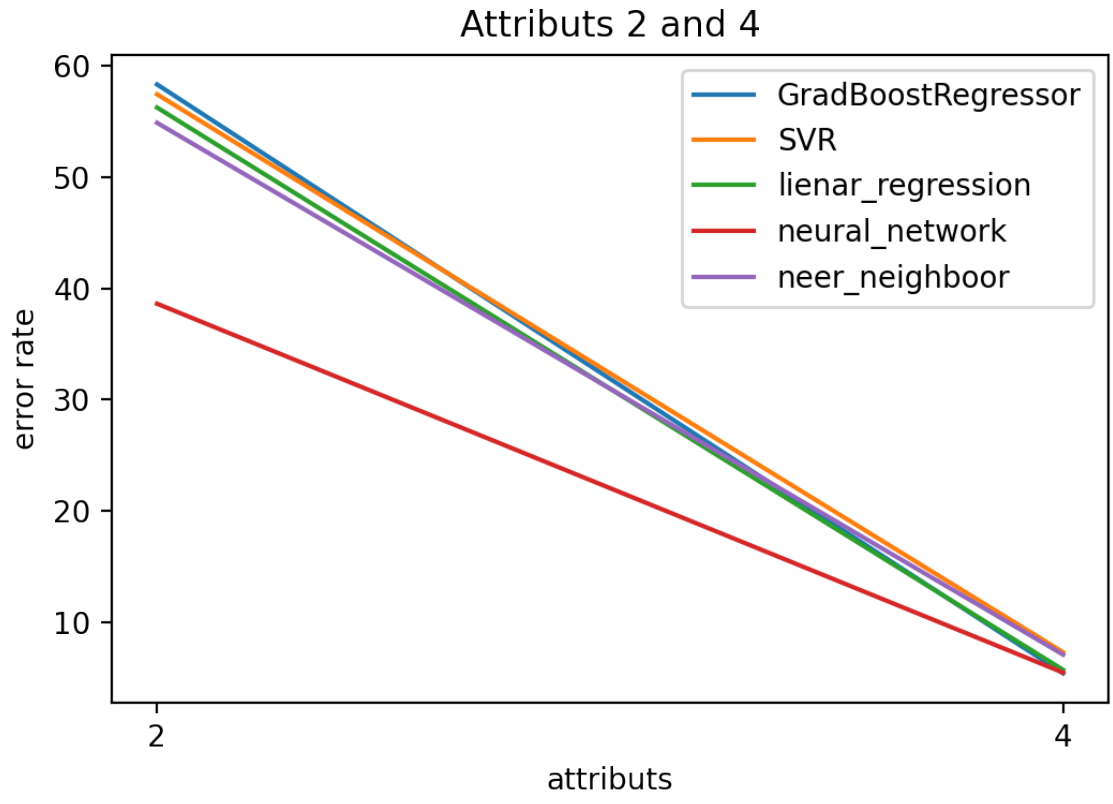
Conclusion

Dans ce travail pratique, nous avons manié le "QM9 dataset" avec pour but de prédire des propriétés moléculaires. Les 4 attributs à prédire était : 'logP', 'RBN', 'MW' et 'RN'. Pour se faire, nous avons choisit différentes méthodes de classement de données : 'Linear Regression', 'KNN - nearest neighboor', 'Gradiant Boosting Regressor', 'SVR', 'Neural Network'. L'idée était pour chaque méthode de tester différents paramètres spécifiques ou non à celle-ci, afin d'optimiser le maximum notre erreur pour chacune des propriétés moléculaires.

Suite à de nombreux tests, nous avons pu obtenir des résultats satisfaisant pour la prédiction des attributs. Nous avons séparé ceux-ci en deux graphes, le premier contenant les attributs : ['logP', 'MW'] avec leur erreur moyenne et le deuxième contenant les attributs : ['RBN', 'RN'] avec leur erreur rate.
En général, le neural network est celui qui performe le mieux pour les attributs ['logP','RBN', 'MW']. Nous observons l'inefficacité du GradiantBoosterRegressor ainsi que du SVR pour l'attribut ['MW']. Ceux-ci peuvent être expliqué par les manque de tests pour obtenir des paramètres optimales. Pour le GBR, on aurait par exemple pu tester le paramètre $n\_estimators$ qui est le nombre d'étapes de boosting à performer.

Finalement, choisir 1 seule méthode ne serait pas idéal, cependant si l'on devait choisir la meilleure méthode serait le neural network au niveau de l'efficacité de prédiction. Concernant l'efficacité au niveau du temps d'excécution, la méthode la plus rapide et efficace est la linear regression. Pour l'attribut ['RBN'], on remarque de très mauvais résultats. Nous ne sommes pas sûre d'où provient l'erreur. Un des problèmes que nous avons rencontré est que n'avons pas de connaissances de le domaine, ce qui aurait pu aider à comprendre les mauvaises prédiction pour certains modèles ou même choisir des méthodes plus adaptées.


Attributs 1 and 3
lienar_regression
neural_network
GradBoostRegressor
SVR
error
0.0
0.2
0.4
0.6
0.8
1
3
attributs


Attributs 2 and 4
GradBoostRegressor
SVR
lienar_regression
neural_network
neer_neighboor
error rate
10
20
30
40
50
60
2
4
attributs