UNIVERSITÉ DE RENNES 1

istic Informatique Électronique

IDM - Projet DSL : Rapport individuel

Hélène Heinlé, M2 MIAGE, DLIS, ISTIC, 2020-2021

I. Contexte

Notre domaine d'étude était celui du Machine Learning Régression (ou ML Régression).
La ML Régression consiste en un ensemble de méthodes d'apprentissage automatique qui nous permettent de prédire une variable continue (Y), en fonction de la valeur d'une ou plusieurs variables prédictives (X).

Afin de faciliter ce processus pour un utilisateur, nous avons créé un langage dédié à ce problème, soit un DSL. Pour spécifier ce DSL, nous avons créé une grammaire assez légère qui permet à un utilisateur de spécifier des fichiers avec l'extension ".mlr". Un tel fichier permet de définir le contexte d'une régression à réaliser. Dans un fichier ".mlr", l'utilisateur peut spécifier :

- le fichier CSV contenant le jeu de données (variable X et variable(s) Y)
- les indices de la variable cible et des variables prédictives dans le CSV
- la taille de l'ensemble de test (nous séparons le jeu de données en deux sous-ensembles)
- l'algorithme à utiliser pour faire la régression
- le type d'erreur à calculer

Voici un exemple de fichier ".mlr" respectant notre grammaire :

```
regression{
   file: "datasets/ozone.csv",
   testSize : 30,
   predictiveVariables : {1,2,9,10},
   targetVariable: 0,
   algorithm: Linear,
   errorType: rmse
}
```

Nous retrouvons les paramètres suivants :

- `file:` chemin vers un fichier CSV qui est le jeu de données (variable X et variable(s) Y)
- `testSize:` la taille de l'ensemble de test en pourcentage. Cette taille représente le pourcentage du jeu de données sur lequel le modèle va être testé. Le reste du jeu de données sert d'ensemble d'apprentissage au modèle de prédiction. Il est nécessaire de renseigner un nombre entier compris entre 0 et 99, la valeur conseillée étant 30%)
- `predictiveVariables:` les indices des variables prédictives dans le fichier précédent. Ici, nous prenons x1, x2, x9 et x10 pour prédire y.
- `targetVariable:` l'indice de la variable cible (y) dans le fichier précédent
- `algorithm:` l'algorithme à utiliser pour faire la régression. Les valeurs acceptées par notre grammaire sont : RegressionTree, SVM ou Linear.
- `errorType:` le type d'erreur à calculer afin de montrer la précision de la prédiction. Les valeurs acceptées par notre grammaire sont : rmse (root mean squared error), mae (mean absolute error) ou r2 (r2 score).

Dans cet exemple, nous travaillons avec le jeu de données “ozone.csv”. Les lignes de ce fichier représentent des jours. Pour chacun de ces jours, il y a plusieurs mesures météorologiques :

- y : la valeur maximale de la concentration en ozone (O3) dans la journée
- x1, x2, et x3 : les températures à 9h, 12h, et 15h
- x4, x5, et x6 : la nébulosité à 9h, 12h, et 15h
- x7, x8, et x9 : projection du vent sur l'axe Est-Ouest à 9h, 12h et 15h
- x10 : la concentration maximale en ozone de la veille

Afin d’obtenir les résultats des régressions, les fichiers “.mlr” sont compilés.
Nous avons choisi de faire un compilateur Python et un compilateur R.
Ces compilateurs, à partir de fichiers “.mlr”, traversent le modèle afin de construire le programme (Python ou R) de régression correspondant. Ils produisent donc en sortie, un fichier “.r” ou “.py” qui est exécuté.
Ces compilateurs permettent aussi d’inscrire les résultats de la régression dans des fichiers CSV (benchmark_Pyth.csv et benchmark_R.csv). Cela nous permet de comparer les performances des deux compilateurs sur des configurations “.mlr” égales.

II. Etudes des compilateurs

Afin de tester nos compilateurs, nous avons créé une classe de test en Xtend qui contient plusieurs cas de tests avec 18 configurations “.mlr” différentes (voir la classe de test Xtend). Ces configurations donnent toujours une seule variable cible mais le nombre de variables prédictives varie. Chaque cas de test fourni un texte respectant notre grammaire à nos deux compilateurs R et Python. Faire ces tests nous a permis de comparer les performances de nos compilateurs sur des modèles de régression à unique ou multiples variables prédictives.

A. Différences fonctionnelles

Nous avons remarqué des différences entre R et Python :

- sur les indices des variables : pour pointer vers une variable étant dans la 1ère colonne du CSV, Python acceptera la valeur 0 alors que R prendra 1. Notre compilateur R augmente les paramètres `predictiveVariables` et `targetVariable` de 1 afin d’avoir des fichiers “.mlr” identiques à passer aux compilateurs.
- sur l’erreur r^2 : la méthode r2 en R ne marche pas pour tous les algorithmes spécifiés dans notre grammaire. Nous avons donc fait un calcul au lieu d’utiliser une méthode de R.

B. Écarts de performance

L'écriture dans les fichiers CSV "benchmark.csv" a permis de voir les écarts de performance entre nos deux variantes. Étant donné le problème sur les indices évoqué au point précédent, vous retrouverez dans les figures suivantes le décalage sur les indices des variables. Cela ne change en rien l'analyse puisque, pour R, un indice 1 indique la première colonne d'un CSV et pour Python 0 indique cette colonne.

Dans le fichier "Ozone_benchmark_R_vs_Pyth.ods" joint au rendu, vous pourrez retrouver les analyses des fichiers CSV montrées dans les figures suivantes suite à une exécution de la classe de test Xtend TestCompilers1. Dans cette classe de test, dans le but de comparer les variantes, nous nous basons sur le même jeu de données "ozone.csv".

Temps d'exécution

Le premier critère que j'ai choisi pour évaluer nos compilateurs est le temps d'exécution. Depuis les fichiers CSV de benchmarks obtenus après une exécution de la classe TestCompilers1, j'ai éclairci et trié les données. Le temps moyen d'exécution moyen pour chaque algorithme et dans chaque langage a été calculé (figure 1 et "Ozone_benchmark_R_vs_Pyth.ods").

Sur la figure 1, nous observons que Python reste plus rapide que R. En R comme en Python, c'est l'algorithme Regression Tree le plus rapide.

Algorithme	Temps d'execution moyen (ms)
regressionTree	0,09475
SVM	0,3831
linear	0,401233333333333

Algorithme	Temps d'execution moyen (ms)
regressionTree	0,30635
SVM	0,220783333333333
linear	0,159233333333333

Figure 1 : Temps d'exécution moyen selon l'algorithme utilisé (jeu de données "ozone.csv"). Gauche : R, droite : Python

Le temps d'exécution varie selon les performances et la disponibilité de la machine utilisée. On peut le voir sur le fichier "Ozone_benchmark_R_vs_Pyth.ods", dans la seconde feuille. Sur les trois exécutions, les temps varient. Le temps d'exécution n'est donc pas un critère suffisant pour évaluer pleinement nos compilateurs.

Précision des prédictions

Le second critère retenu pour la comparaison des compilateurs a été la précision de la régression effectuée.
A l'écriture d'un fichier ".mlr", l'utilisateur à la possibilité de choisir entre trois calculs d'erreur :

- erreur quadratique moyenne (ou root mean squared error, ou rmse) : mesure fréquemment utilisée pour voir les différences entre les valeurs prédites par un

modèle de prédiction et les valeurs observées. Ces écarts sont appelés résidus. La rmse vaudra 0 pour un modèle parfait qui prédit exactement la vérité.

- erreur absolue moyenne (ou mean absolute error ou mae) : la mae mesure l'ampleur moyenne des erreurs dans un ensemble de prédictions, sans tenir compte de leur direction. La mae vaudra 0 pour un modèle parfait qui prédit exactement la vérité.
- le coefficient de détermination R^2 (ou r2 score) est une mesure de la qualité d'une prédiction. Un R^2 de 1 montre un modèle parfait, qui prédit la vérité.

Dans le fichier "Ozone_benchmark_R_vs_Pyth.ods" et sur la figure 2, on observe que, pour "ozone.csv" :

- en R : en moyenne, c'est l'algorithme Regression Tree qui permet d'avoir une meilleure précision
- en Python : en moyenne, c'est l'algorithme linéaire qui permet d'avoir une meilleure précision

	Erreur moyenne		
Algorithme	RMSE	MAE	R2
linear	19,0548564791453	15,1081814579228	0,535009055534601
SVM	20,1294251443611	17,7955730549731	0,457261376320434
regressionTree	17,4476028815729	16,7282120395328	0,723534663751051

	Erreur moyenne		
Algorithme	RMSE	MAE	R2
linear	17,3821434121865	14,1181539256239	0,374767013282914
SVM	25,319715473962	18,4212043846778	0,128578044725753
regressionTree	19,7235731940753	14,747311827957	0,048678408794786

Figure 2 : Erreurs moyennes (jeu de données "ozone.csv"). Haut : R, bas : Python

Pour le jeu de données "ozone.csv", on voit sur la figure 3 que les 2 compilateurs sont d'accord sur les modèles de prédiction les plus précis :

- en terme de rmse : x1, x2, x9 et x10 en regression tree
- en terme de mae : x1, x2, x9 et x10 en linéaire

Par contre :

- en termes de R2 : x1 en SVM pour R et x1 en regression tree pour SVM. On peut se demander si R2 est une bonne mesure d'erreur pour comparer des modèles différents. R2 montre le taux de données expliquées par le modèle. La valeur du R2 est dépendante de l'étendue de la variable prédictive

R			
Algorithme	Variable(s) predictrice(s)	Type Erreur	Erreur
SVM	[2]	r2_score	0,359863
linear	[2]	r2_score	0,497161
SVM	[2, 3, 10, 11]	r2_score	0,554659
linear	[2, 3, 10, 11]	r2_score	0,572857
regressionTree	[2]	r2_score	0,648485
regressionTree	[2, 3, 10, 11]	r2_score	0,798584
linear	[2, 3, 10, 11]	mean_absolute_error	11,950573
regressionTree	[2, 3, 10, 11]	mean_absolute_error	14,376344
SVM	[2, 3, 10, 11]	mean_absolute_error	14,895194
regressionTree	[2, 3, 10, 11]	root_mean_squared_error	15,524929
SVM	[2, 3, 10, 11]	root_mean_squared_error	17,480427
linear	[2]	mean_absolute_error	18,265790
linear	[2, 3, 10, 11]	root_mean_squared_error	18,293588
regressionTree	[2]	mean_absolute_error	19,080080
regressionTree	[2]	root_mean_squared_error	19,370277
linear	[2]	root_mean_squared_error	19,816125
SVM	[2]	mean_absolute_error	20,695952
SVM	[2]	root_mean_squared_error	22,778423

Figure 3 : Comparaison de l'exécution des compilateurs sur 18 configurations et un jeu de données "ozone.csv". Haut : R, bas : Python

Python			
Algorithme	Variable(s) predictrice(s)	Type Erreur	Erreur
regressionTree	[1]	r2_score	-0,262521
linear	[1]	r2_score	0,079409
SVM	[1]	r2_score	0,085619
SVM	[1, 2, 9, 10]	r2_score	0,171537
regressionTree	[1, 2, 9, 10]	r2_score	0,359877
linear	[1, 2, 9, 10]	r2_score	0,670125
linear	[1, 2, 9, 10]	mean_absolute_error	12,610303
regressionTree	[1, 2, 9, 10]	mean_absolute_error	13,548387
regressionTree	[1, 2, 9, 10]	root_mean_squared_error	14,128443
linear	[1]	mean_absolute_error	15,626005
regressionTree	[1]	mean_absolute_error	15,946237
linear	[1, 2, 9, 10]	root_mean_squared_error	17,043435
linear	[1]	root_mean_squared_error	17,720852
SVM	[1, 2, 9, 10]	mean_absolute_error	17,900003
SVM	[1]	mean_absolute_error	18,942406
SVM	[1, 2, 9, 10]	root_mean_squared_error	23,203919
regressionTree	[1]	root_mean_squared_error	25,318703
SVM	[1]	root_mean_squared_error	27,435512

C. Conclusion sur les performances

J'ai fait cette analyse par un logiciel de calcul plusieurs fois sur le même jeu de données et également en se basant sur d'autres jeux de données comme "dataset.csv" (voir TestCompilers2). Les résultats changent, ce qui est logique, mais les conclusions à tirer restent les mêmes.

Au niveau de la réalisation d'une régression :

- R est plus fonctionnel alors que Python est plus orienté objet : avec des fonctions comme lm, predict et d'autres, R laisse les fonctions faire le travail. Python utilise des classes comme LinearRegression pour faire le modèle de prédiction.
- R intègre plus de fonctionnalités d'analyse de données alors que Python s'appuie sur des paquets comme pandas, scikit-learn et autres. Cela à une répercussion sur l'implémentation de Docker : j'ai remarqué en essayant de faire un Dockerfile pour Python, que la construction de l'image était un peu plus longue sur ma machine.

Au niveau de la performance d'une régression :

- Python est plus rapide que R pour faire de la régression.
- Nos compilateurs sont faits pour calculer une prédiction depuis des variables données par l'utilisateur qui rédige un ".mlr". En général, avant d'effectuer une régression, nous analysons quelle est la meilleure combinaison de variables prédictives pour prédire le plus fidèlement possible la variable cible. Ici, il faut assumer que le choix fait par l'utilisateur est correct. Si l'erreur de prédiction à la fin indique un résultat moyen, ce n'est pas la faute du compilateur mais plutôt de la configuration choisie. Nos deux compilateurs sont plutôt d'accord sur les erreurs de précision pour un même jeu de données.

IV. Critique de notre DSL

Nous avons déjà eu l'occasion de faire un peu de régression durant notre cursus MIAGE mais nous étions loin d'être des experts. Faire ce projet et cette étude m'a permis de voir les améliorations que nous pourrions apporter à notre grammaire :

- Nos compilateurs utilisent la séparation en deux ensembles de test et d'apprentissage afin de faire apprendre le modèle de prédiction et de tester sa justesse. Cette répartition (généralement 30% test et 70% apprentissage), est faite aléatoirement : les lignes du jeu de données données sont répartis aléatoirement dans l'un ou l'autre des ensembles. Pour des résultats plus juste, nous aurions pu supprimer la variable aléatoire.
- Afin de mieux comparer nos compilateurs, notre grammaire aurait pu ne pas donner la possibilité de choisir un type d'erreur. Ainsi, nos compilateurs auraient calculé trois types d'erreurs sur un même modèle de prédiction.

Remarque : Le jeu de données "dataset.csv" contient 30 colonnes (des films) et 1200 lignes (les notes attribuées à ces films par les utilisateurs) de la plateforme. La variable à prédire ici est la 30 ème colonne (le film "F30") c'est-à -dire qu'il faut estimer la note qui sera attribuée par les utilisateurs à ce film "F30". Les variables prédictives sont toutes les notes attribuées précédemment à d'autres films.

V. Reproduction

Pour reproduire les régressions que nous avons faites, il faut lancer une classe de test Xtend qui spécifie 18 configurations ".mlr". Ces classes de test s'appuient sur ces jeux de données.
Les résultats des régressions qui vont permettre de comparer les deux compilateurs sont écrits dans ce dossier.
Afin d'analyser plus en profondeur ces données, mes fichiers "Ozone_benchmark_R_vs_Pyth.ods" ou "Dataset_benchmark_R_vs_Pyth.ods" joints au mail de rendu peuvent être utilisés. Il contiennent chacun trois feuilles :

- La première permet d'afficher les résultats contenus ici.
- La seconde permet de voir les temps d'exécution moyens grâce à des formules (pour "Ozone_benchmark_R_vs_Pyth.ods", il y a trois exécutions).
- La troisième sert à analyser les erreurs de précision.