IDM - Rapport d'observation sur les compilateurs

Enseignant : Mathieu ACHER

Emile GEORGET – M2 MIAGE parcours DLIS

ISTIC - Université Rennes 1 - 2020 - 2021

Sommaire

I. Contexte

Notre objectif est de réaliser un apprentissage par régression sur des jeux de données au format CSV via un DSL qui nous permet de spécifier de façon « simple » les paramètres de cette régression. Ce DSL peut être compilé vers deux langages différents que nous avons choisi pour leurs fonctionnalités et leurs bonnes aptitudes à réaliser des calculs mathématiques : R et Python.

Pour rappel, un DSL est un langage dont les propriétés sont adaptées à un domaine particulier, dans le but d'être utilisé par un non informaticien. Par exemple dans notre cas, le langage est adapté à l'usage par les « statisticiens » et est inspiré de la structuration d'un fichier JSON qui est relativement simple à appréhender.

Un apprentissage par régression a pour objectif de trouver une corrélation entre une variable cible et un certain nombre variables prédictives pour créer un modèle de prédiction qui pourra être utilisé sur un plus grand nombre de valeurs par la suite.

Exemple d'une ligne de code du DSL :

regression {file: "datasets/ozone.csv", testSize: 30, predictiveVariables: {2,3,10,11}, targetVariable: 1, algorithm: Linear, errorType: rmse}

file: Prend en paramètre le chemin vers le jeu de données CSV. Il doit être entouré de guillemets

testSize: taille de l'ensemble de test en %. Le jeu de données va être découpé en deux parties contenant des membres aléatoires. La taille de l'ensemble de test représente le pourcentage du jeu de données sur lequel le modèle de prédiction doit être testé. Le pourcentage restant constitue l'ensemble d'apprentissage qui permettra d'entraîner le modèle de prédiction.

(!) Il est indispensable de renseigner un nombre entier compris entre 1 et 99 pour faire fonctionner le programme, nous vous conseillons de rester à une valeur de 30% par défaut.

predictiveVariables: le ou les variable prédictives. Il est nécessaire d'entourer cette ou ces variables d'accolades.

targetVariable: la variable cible (ne pas mettre d'accolades)

algorithm: l'algorithme utilisé pour la régression. Nous prenons en charge trois algorithmes : Linear, RegressionTree et SVM (écrire un de ces trois noms).

errorType : le type d'erreur calculé (utilisé pour donner une idée sur la précision et l'efficacité du modèle de prédiction). Ecrire mae, r2 ou rmse.

Nous allons étudier les résultats de nos différents compilateurs et nous comparerons les deux sur deux critères : le temps d'exécution et la précision.

II. Etude des compilateurs

a. Prérequis

Il faut noter que les deux compilateurs ont été comparés sur la base d'instructions DSL identiques mais que les ensembles d'entrainement dans dans les jeux de données sont de taille définie avec des valeurs sélectionnées aléatoires. Cela induit donc obligatoirement un biais dans la comparaison. Les tests ont été effectués sur un jeu de données contenant des mesures de la concentration d'ozone. Vous pouvez également refaire ces tests avec un autre jeu de données qui contient des notes attribuées à des films par le public.

b. Vitesse d'exécution

Le premier paramètre que nous allons étudier est la vitesse d'exécution. Il n'est pas forcément le plus intéressant à étudier dans le cadre de la régression car du biais est induit par la machine, le jeu de données, le calcul d'erreur…

Nous avons étudié le temps d'exécution moyen d'une instruction de notre DSL avec compilation en R et en Python.

Notre protocole est de tester 3 fois chaque teste pour les deux compilateurs. On fait une moyenne du temps d'exécution de ces trois exécutions, pour chaque test et chaque compilateur.

Les tests incluent l'ensemble des combinaisons algorithmes/calcul d'erreur possibles pour essayer d'avoir une variété d'instructions.

Les tests incluent également des instructions avec plusieurs variables prédictives, d'autres avec des variables prédictives uniques.

Nous avons ensuite fait la moyenne des moyennes de temps d'exécution de chaque instruction, pour chaque compilateur. L'objectif étant de définir que compilateur sera le plus pertinent en cas d'exécution d'un grand nombre d'instructions ou de jeux de données volumineux.

Langage cible	Durée d'exécution moyenne instruction (ms)
R	1,051
Python	0,184

La compilation en Python, selon nos résultats semble donc près de six fois plus rapide en moyenne que la compilation en R.

Cela s'explique certainement par des différences inhérentes aux langages eux-mêmes.

On préfèrera donc, en général, Python à R si le critère principal est la vitesse d'exécution, notamment si les jeux de données sont importants.

L'analyse de la durée d'exécution par algorithme pour un même compilateur n'est pas très pertinente car elles sont relativement proches. Néanmoins les données sont disponibles dans le tableur fourni.

c. Justesse des prédictions

Il est intéressant de s'intéresser à la justesse des prédictions. Cela permet d'avoir une idée sur l'intérêt et la fiabilité du modèle de prédiction, qui dépend lui-même de l'algorithme de prédiction choisie.

Nous avons choisi d'implémenter trois différentes méthodes pour mettre en valeur la justesse ou non d'une prédiction.

- MAE (Mean Absolute Error ou Erreur Moyenne Relative) : sert pour déterminer l'ordre de grandeur de l'erreur du modèle. Plus ce nombre est petit, plus le modèle est fiable.
- RMSE (Root Mean Squared Error ou Racine Carré de l'Erreur Moyenne) : même utilité que MAE. Cependant, RMSE donne un plus grand poids aux grosses erreurs et est donc plus souhaitable quand les erreurs de grand calibre sont problématiques. Plus ce nombre est petit, plus le modèle est fiable.
- R2 est utilisé pour déterminer si le modèle suit la tendance du jeu de données. Il représente un pourcentage de variables expliquées par le modèle. Plus le pourcentage est élevé, plus le modèle explique de données, mais cela ne veut pas dire qu'il est fiable. R2, en dehors d'un algorithme de régression linéaire, peut être négatif. Dans ce cas, cela veut dire que le modèle ne suit pas la tendance des données ce qui indique potentiellement que le modèle n'est pas correct.

On commence par regarder quel langage explique le mieux les données, en moyenne.

Nous avons exécuté chaque test une fois pour chaque algorithme. Les instructions sont les mêmes que pour les tests de temps d'exécution. Des données plus complètes sont fournies dans le tableur fourni.

Langage cible	% moyen de données expliquées (R2)
R	0,460
Python	0,414

On remarque que R explique, en moyenne, plus de données que Python. Cependant, il n'y a pas une grande différence.

Nous allons regarder pour quel algorithme chaque compilateur s'en sort le mieux dans l'explication des données :

Langage cible	Meilleur algorithme (en moyenne)	% moyen de données expliquées (R2)
R	Regression Tree	0,552
Python	Regression Tree	0,515

Encore une fois, R explique un peu mieux les données que Python, mais cette différence est négligeable.

On peut donc dire que les deux algorithmes offrent des modèles qui explique de façon similaire les données.

Cependant, le pourcentage moyen de données expliquées par le meilleur algorithme est tout juste correct (un peu plus de 50% des données).

Cela peut s'expliquer par le fait que les variables prédictives utilisées dans les instructions ne sont pas forcément les plus optimales. De plus, l'ensemble d'entrainement du modèle étant de taille fixe mais de valeurs aléatoires, cela peut jouer dans la quantité de données expliquées.

Regardons maintenant la fiabilité des prédictions. Nous utiliserons dans notre cas l'indicateur RQSE qui met plus en relief les erreurs importantes.

Langage cible	Moyenne des erreurs quadratique moyenne sur le taux d'ozone (RQSE)
R	17,402
Python	22,959

On remarque que les modèles de prédictions fournis par R sont en moyenne plus fiables que ceux de Pythone. L'écart moyen est faible mais non négligeable.

Concentrons-nous maintenant sur l'algorithme le plus fiable en moyenne pour chaque compilateur :

Langage cible	Meilleur algorithme (en moyenne)	Moyenne des erreurs quadratique moyenne sur le taux d'ozone (RQSE)
R	SVM	16,157
Python	LINEAR	18,135

On remarque que pour le compilateur R, SVM est l'algorithme le plus fiable en moyenne alors qu'en Python, l'algorithme LINEAR est plus fiable en moyenne.

L'aléatoire dans la séparation des jeux de données peut expliquer ces différences, mais il est également possible que l'algorithme SVM pour R et l'algorithme LINEAR pour Python, soient plus efficaces que les autres algorithmes qu'ils prennent en charge.

On ne peut cependant pas recommander par défaut un de ces deux algorithmes car nous ne les avons pas encore comparés avec leur homonyme de l'autre compilateur.

Nous allons donc nous poser la question s'il y a des différences significatives de fiabilité entre deux mêmes algorithmes de compilateurs différents.

On remarque qu'il y a une différence notable entre le RQSE de R et de Python sur l'algorithme SVM.

En effet, le SVM de Python semble être vecteur, en moyenne de plus de 71% d'erreurs supplémentaires que le SVM de R. L'aléatoire de séparation du jeu de données semble de pas totalement expliquer cette différence, car elle est nettement supérieure aux écarts entre les autres algorithmes. Cela suggère qu'un des deux compilateurs doit mal utiliser cet algorithme, soit R indique une erreur trop faible, soit Python indique une erreur trop élevée.

On recommandera donc d'éviter SVM dans le cas où le taux d'erreur doit être bas, même si c'est le meilleur pour R, car on peut se demander si R ne « sous-estime » pas les erreurs.

L'algorithme LINEAR semble plus équilibré avec une variation des résultats entre compilateurs de 3.5% seulement.

III. Conclusion

R n'est pas un langage à recommander pour une compilation rapide mais il semble induire moins d'erreurs de prédictions que Python. Néanmoins, si on compare les résultats de certains algorithmes de régression entre les deux compilateurs, on peut se demander s'il n'existe pas des différences dans la méthode de calcul pour un même algorithme, de compilateurs différents.

Il est donc important de comprendre que les résultats dont nous disposons peuvent induire du biais, soit lié au langage utilisé, soit lié aux données, soit lié à la machine (pour le temps d'exécution). Ce biais pourrait être réduit en améliorant les données testées. Il faudrait s'assurer que la séparation des données pour les deux compilateurs, soit identique entre eux pour chaque exécution.