Exercices - Semaine 2

Raphaël Nedellec

Préparation

1. Vérifiez que les options globales de RStudio sont conformes aux options recommandées dans le cours.
2. Créez un projet RStudio intitulé `cours_r_semaine_2`. Ce projet sera créé vide. Les options utilisation de renv et de git peuvent être ignorées.
3. Créez un dossier data au sein du projet. Téléchargez le .zip correspondant à la semaine 2 sur le site du cours et le décompresser dans le dossier data.

Exercice 1 : Régression linéaire - classe S3 et fonctions génériques

Le but de cet exercice est de manipuler des fonctions et d'implémenter manuellement quelques fonctions de R utiles pour faire de la modélisation statistique.

Un modèle de régression linéaire multiple s'écrie sous la forme $Y = \beta_0 + \beta_1 X_1 + ...\beta_n X_n + \epsilon$, où les ϵ sont indépendants, d'espérance nulle et de variance constante σ^2.

1. Lire le fichier `exo_1_2.csv`. Décrire le fichier de données.
2. Construire un ensemble d'apprentissage `data_train` contenant les 500 premières lignes du dataset. `data_test` sera le jeu de validation et comprendra les lignes restantes.
3. On cherche à expliquer le prix des biens immobiliers en fonction des autres variables explicatives du dataset. Utiliser la fonction `lm` pour entrainer un modèle de régression linéaire sur le jeu de données d'apprentissage en régressant le prix sur le reste des variables explicatives. Décrivez l'objet de modèle obtenu. Quels sont les paramètres obtenus ? Quelle est la nature de l'objet obtenu ? Comment sont incluses les variables `mainroad`, `basement`, `hotwaterheating`, `airconditioning`, `prefarea` et `furnishingstatus`dans le modèle ?

Dans la suite de l'exercice, nous chercherons à implémenter notre propre algorithme de régression linéaire.

4. Estimer les beta minimisant l'erreur quadratique $||Y - \beta X||^2$ par la méthode des moindres carrés revient à calculer $\hat{\beta} = (X'X)^{-1}X'Y$. Implémenter l'estimation par moindres carrés en développant une fonction intitulée `mco`, prenant en entrée deux paramètres, y, et X. La fonction retournera le vecteur de poids `beta`.

- indice : on pourra s'intéresser aux fonctions `?solve` et `crossprod`. L'opérateur de produit matriciel est l'opérateur `%*%`.

5. Usuellement, les modèles de régression linéaires incluent une constante. Concrètement, cela se traduit par l'introduction d'une colonne constante de 1 dans la matrice de design. Construire une fonction `add_constant` qui ajoute à la matrice de variables explicatives une colonne constante.
6. Pour des raisons d'identifiabilité, il faut traiter avec attention les variables catégorielles. Il faut définir des contrastes. Une variable catégorielle à n modalités va être transformées en n-1 variables. Définir une fonction `mat_factor` qui prend en entrée un vecteur de type facteur à n modalités et renvoie en sortie un matrice de n-1 colonnes. Par exemple, le vecteur : `as.factor(c("a", "b", "b", "c"))` sera transformée en la matrice `matrix(c(0, 1, 1, 0, 0, 0, 0, 1), ncol = 2L, dimnames = list(NULL, c("b", "c")))`.
7. Pour que le problème ait une solution, il faut que la matrice X soit inversible. Implémenter la fonction `my_lm` prenant en entrée un vecteur y (variable cible) et une data.frame X (matrice des variables explicatives).

- il faudra tester que la taille de y a la même taille que le nombre de lignes de X ou lancer une erreur ;
- transformer la data.frame X en matrice de design; chaque variable catégorielle doit être transformée en utilisant la fonction `mat_factor` ;
- s'assurer que la constante est bien dans le modèle ;
- vérifier que la matrice de design est inversible ou stopper les exécutions en lançant une erreur ; on pourra utiliser la fonction `rankMatrix` du package Matrix.
- utiliser la fonction définie en 4. pour estimer les beta.

En sortie de la fonction, on retournera une liste contenant les informations suivantes :

```r
list(
  coef = ..., # vecteur nommés des coefficients
  residuals = ..., # résidus (Y - beta*X),
  fitted.values = ..., # beta*X
  y = ... # vecteur y original
)
```

5. Modifier la fonction `my_lm` pour que l'objet retourné soit de classe `"my_lm"`.

6. Implémenter la fonction générique `predict.my_lm` qui prendra deux arguments `object` héritant de la classe `my_lm` et un argument `newdata` de type data.frame. Cette fonction renvoie en sortie un vecteur $\hat{y}$ de prévision correspondant au produit βX^{new}.
7. Implémenter la fonction `print.my_lm` générique de telle sorte que les coefficients obtenus pour les différentes variables, ainsi que la comparaison entre les 10 premières valeurs de Y et les 10 premières `fitted.values` soient affichées de manière claire dans la console.
8. Comparez les résultats obtenus en 3. avec les résultats obtenus avec votre fonction `my_lm`. Sont-ils identiques ? Les coefficients estimés sont ils les mêmes ? Obtenez-vous les mêmes prédictions sur le jeu de test `data_test` ?
9. Utilisez la fonction `microbenchmark` de la librairie `microbenchmark` pour mesurer le temps d'exécution de votre fonction et celui de la fonction lm. Commentez et analysez les différences.

Exercice 2 : Fonctionnelles

Dans cette exercice, nous allons travailler avec les fonctionnelles : des fonctions qui prennent en argument des fonctions. Nous travaillerons à nouveau avec le jeu de données de l'exercice 1.

1. Lire le fichier `exo_1_2.csv` et l'affecter à la variable `data_exo_2`.
2. On souhaite tout d'abord calculer quelques statistiques pour les biens conditionnellement au nombre de salles de bains. Utiliser la fonction `split` pour séparer le dataset complet en autant de datasets que de modalités de la variable `bathrooms`. Que fait la fonction `split` ? On affectera le résultat à la variable `data_split`. Décrire.
3. En utilisant la fonction `lapply`, calculez le prix minimal, médian, moyen, et maximal des biens par nombre de salles de bain. Comparez les résultats en utilisant `simplify = TRUE` puis `simplify = FALSE` comme argument de la fonction `lapply`.
4. Utiliser `sapply`, en lieu et place de `lapply`. Expliquez la différence.
5. Nous savons que pour chaque catégorie de biens, nous calculons en sortie un vecteur de 4 valeurs numériques (prix minimal, médian, moyen, et maximal). Utilisez la fonction `vapply` en lieu et place de `sapply`. Quel avantage y a-t-il à utiliser `vapply` ?
6. Executez la commande suivante : `data_split[[4L]] <- data_split[[4L]][, -1]`. Que fait cette commande ?
7. Refaites tourner le code de la question 4. Quels sont les nouveaux résultats, en particulier pour les biens avec 4 salles de bain ?
8. Modifier votre fonction pour qu'une erreur soit explicitement déclarée si chaque sous-jeu de données ne contient pas de colonne intitulée `price`. En exécutant votre code à nouveau, que se passe-t-il désormais ?
9. Utilisez la fonction `tryCatch` pour capturer l'erreur et renvoyer `NA` si la colonne `price` est manquante.
10. On s'intéresse désormais à nouveau au dataset originel `data_exo_2`. On souhaite calculer l'indicateur précis suivant :

- si `mainroad == TRUE` et `prefarea == TRUE`, alors `indicateur <- price/area + 1000`
- sinon, `indicateur <- price/area - 5000 + bedrooms*100`

Utilisez la fonction `mapply` pour le faire. 10. Répondez à l'intégralité des questions à nouveaux en utilisant les fonctions `map_\*` et `pmap_\*` adaptées de la librairie `purrr`. Quels sont les différences entre les fonctions de `purrr` et les fonctions de base ? Quels sont les avantages de `purrr` ?

Exercice 3 : Advent of code : algorithmes et fonctions.

1. Lire le fichier `exo_3_ex.txt`. Décrivez son contenu et affecter le à la variable `data_3_ex`.
2. Chaque valeur de `data_3_ex` correspond à un arbre, et cette valeur représente la hauteur de l'arbre. On considérera que cet ensemble d'arbres représente une forêt. On cherche à savoir quels sont les arbres visibles depuis l'extérieur de la forêt. Un arbre est visible si et seulement si tous les arbres le séparant d'un bord de la forêt sont de taille strictement inférieure à la sienne. Implémenter 4 fonctions `est_visible_nord`, `est_visible_sud`, `est_visible_est`, `est_visible_ouest` qui détermine si chacun des arbres est visible depuis un des 4 bords de la forêt.
3. Quels sont les arbres qui ne sont pas visibles depuis l'extérieur de la forêt ? Implémentez une fonction pour faire le calcul.
4. Lisez le fichier `exo_3_full.txt`. Combien d'arbres sont visibles depuis l'extérieur de la forêt dans le jeu de données complet ?
5. On cherche désormais à identifier l'arbre de la forêt qui a la meilleure vue ! Il faut tout d'abord compter le nombre d'arbres visibles dans chaque direction pour tous les arbres de la forêt. Dans cet exemple :

```
30373
25512
65332
33549
35390
```

l'arbre tout en haut à gauche, de hauteur 3, ne voit aucun arbre en direction du Nord (il est au bord), aucun arbre en direction de l'Ouest (il est au bord), 2 arbres en direction du Sud (2, puis 6). L'arbre de taille 6 étant de taille supérieure ou égale à la sienne, sa vue devient bloquée. De même, il voit deux arbres en direction de l'Est (0 et 3).

De la même façon, le deuxième arbre sur la diagonale, de taille 5, voit respectivement 1 arbre dans chacune des directions.

Implémenter 4 fonctions `compter_nb_arbres_visibles_nord` (resp. sud, est, ouest) et comptez pour chaque arbre le nombre d'arbres visibles.

6. Le meilleur arbre est celui dont le produit des scores est le plus élevé. Quel est l'arbre idéal ? Implémentez une fonction pour faire le calcul.
7. Faites le même calcul pour le jeu de données complet `exo_3_full.txt`. Quel est le meilleur score possible ?