Maxence Liogier
Souraya Ahmed
Yacine Ayachi

Régression Logistique Multinomiale

Rapport

Élèves :
Souraya AHMED
Maxence LIOGIER
Yacine AYACHI

Enseignant :
Ricco RAKOTOMALALA

12 décembre 2024

Maxence Liogier
Souraya Ahmed
Yacine Ayachi

Table des matières

1 Introduction

Dans le cadre de notre Master 2 Statistique et Informatique pour la Science des Données (SISE) à l'Université Lumière Lyon 2, nous avons réalisé un projet visant à développer un package R dédié à la régression logistique multinomiale.

L'objectif de ce package est de réaliser de la classification supervisée via une régression logistique multinomiale se basant sur un algorithme de descente de gradient. Ce package peut prendre en compte des variables explicatives mixtes (quantitatives et qualitatives). L'implémentation suit une approche orientée objet, en s'appuyant sur des classes R6.

Ce package est disponible sur GitHub via le lien suivant : `https://github.com/maxenceLIOGIER/regression_logistique`. Une interface interactive développée avec R Shiny a également été développée pour faciliter l'accès et l'utilisation des fonctionnalités proposées.

Ce rapport présente les différentes étapes de notre démarche, de la conceptualisation des algorithmes à leur implémentation et évaluation. Il détaille également les choix méthodologiques réalisés, les outils utilisés et les résultats obtenus.

2 Organisation du package

Notre package R, intitulé `LogisticRegression`, permet de réaliser des régressions logistiques multinomiales. L'architecture du package est organisée comme suit :

- `man/` : Ce dossier contient la documentation des fonctions du package.
- `R/` : Ce dossier contient les scripts et les fonctions.
- `test/` : Contient un script de test du package.
- `NAMESPACE` : Liste des méthodes et fonctions pouvant être utilisées par l'utilisateur.
- `README.md` : Description du projet, présentée sur la page d'accueil du github
- `DESCRIPTION` : Contient les métadonnées du package.
- `LICENSE` : Licence d'utilisation

Ainsi, ce package suit les règles conventionnelles de construction d'un package R.

2.1 Architecture du Programme

Le programme s'organise en plusieurs scripts séparés les uns des autres. Dans cette partie, nous allons présenter ces scripts et leurs relations.

2.1.1 main.R

Ce script est le centre du projet. C'est lui qui définit la classe `LogisticRegression` permettant de mettre au point toutes les méthodes que va appeler l'utilisateur.
Ensuite, toutes les fonctions servant de base à ces méthodes sont consignées dans des scripts distincts. Voici les scripts et leur(s) fonction(s) correspondante(s) :

2.1.2 reg_multinomiale.R

Consigne la fonction `reg_multinomiale()`, qui exécute une régression logistique multinomiale pour estimer les paramètres optimaux. Chaque classe à prédire est traitée à la manière d'un "one-vs-all". Elle fait appel à la fonction `descente_gradient()`.

2.1.3 descente_gradient.R

Consigne la fonction `descente_gradient()`, qui applique l'algorithme de descente de gradient pour optimiser les paramètres d'un modèle de régression logistique, avec une possibilité d'ajouter une régularisation (L1, L2 ou ElasticNet). L'optimisation des paramètres se fait via une boucle. Le nombre d'itérations de cette boucle est définie par l'utilisateur lors de la construction du modèle.

2.1.4 p_values.R

Consigne trois fonctions : `hessienne()`, `calcul_p_values()` et `print_coeffs()`. Ces fonctions permettent lors de l'affichage du modèle de calculer puis d'afficher les p-values, erreurs standard et z scores associés à chaque coefficient de la régression. Cela permet de donner un indicateur permettant de discriminer les variables participant significativement au modèle.
Cette étape est très gourmande car elle passe par l'inversion de la matrice hessienne. C'est pourquoi elle n'est pas réalisée si X a une taille supérieure à 10000 lignes.

2.1.5 prepare_x.R

Consigne deux fonctions : `type_variable()` et `prepare_x()`. La deuxième permet de préparer les variables prédictives afin de pouvoir les utiliser dans la régression. Cela permet de gérer les variables mixtes (qualitatives et quantitatives). Pour cela, les variables quanti sont centrées réduites pour permettre la comparabilité des coefficients affichés par le modèle. Les variables quali sont traitées via un encodage 0-1. `type_variable()` permet de déterminer si une variable est quali ou quanti.

2.1.6 predict_proba.R

Consigne la fonction `predict_proba()`, qui calcule les probabilités d'appartenance de chaque individu à des classes cibles en utilisant les coefficients appris par la régression. Cette fonction est aussi une méthode reprise dans la classe `LogisticRegression`.

2.1.7 calcul_metriques.R

Consigne deux fonctions : `calcul_log_likelihood()` et `calcul_aic()`. Ceci permet de calculer des métriques permettant de donner dees indicateurs de performance du modèle : la log-vraisemblance et le critère d'information d'Akaike.

2.2 Présentation des Méthodes

Après avoir décrit tous les scripts et leur organisation, il convient de présenter un peu plus la classe "LogisticRegression". Pour cela, nous allons présenter toutes les méthodes pouvant être appelées par l'utilisateur et qui composent la classe.

2.2.1 Constructeur de classe : initialize()

Cette méthode sert à initialiser les coefficients. Elle s'appelle de cette manière :
LogisticRegression$new(nb_iters = 500, alpha = 0.01, penalty = NULL, lambda = 0, l1_ratio = 0).
Où nb_iters est le nombre d'itérations réalisées par l'algorithme de descente de gradient.

alpha est le taux d'apprentissage à chaque itération. penalty est la pénalité que l'on peut appliquer à l'algorithme : "l1" = lasso, "l2" = ridge et "elasticnet". lambda est le paramètre de régularisation, toujours positif, plus il est élevé plus la régularisation sera forte. l1_ratio est le ratio de l1 pour elasticnet, compris entre 0 et 1.

2.2.2 Apprentissage du modèle : fit(X, y)

Entraine le modèle de régression logistique multinomiale sur les données d'entrée en utilisant la descente de gradient. X est la matrice des variables prédictives et y est le vecteur des étiquettes à prédire.
C'est cette fonction qui permet de charger les coefficients de la régression (theta), les p-values (dict_coeff) ainsi que les métriques. C'est l'étape la plus gourmande en ressources de calcul.
Cette fonction renvoie un nouvel objet. C'est pourquoi il convient de l'appeler de la manière suivante : trained_model <- model$fit(X, y).

2.2.3 Probabilité selon chaque classe : predict_proba(X, theta)

Calcule les probabilités d'appartenance de chaque observation à chaque classe à partir des coefficients entraînés.
Cette fonction a déjà été présentée ci-dessus car elle a été externalisée de la classe. Nous avons fait ce choix car cette fonction sert pour en dehors de la classe (dans `calcul_metriques.R`).

2.2.4 Prédiction : predict(X)

Prédit la classe d'appartenance pour chaque observation en se basant sur la probabilité maximale. Elle renvoie un vecteur des classes prédites.

2.2.5 Evaluation des performances : test(y_true, y_pred, confusion_matrix=FALSE)

Évalue les performances du modèle à l'aide de métriques standard (précision, rappel, F1-score, matrice de confusion). La matrice de confusion est renvoyée si on la définit comme TRUE.

2.2.6 Informations succinctes : print()

Affiche les coefficients du modèle sous forme d'une liste, séparée par classe. Lorsque l'on est dans un cas binaire, seule la deuxième classe est affichée (la première donnant des coefficients opposés à la deuxième et pouvant donc être déterminée facilement) car la classe y=+ ou y=1 est la plus intéressante.
Pour plus d'informations, il faut utiliser la fonction `summary()`.

2.2.7 Informations détaillées : summary()

Affiche, comme `print()`, les coefficients du modèle par classe mais également un résumé des métriques du modèle (log-vraisemblance, AIC) et, si possible (si <10000 lignes), les p-values des coefficients.

2.2.8 Importance des variables : var_importance(graph=TRUE)

Évalue l'importance de chaque variable (hors intercept). Le calcul de l'importance est basé sur les coefficients absolus moyens. Ceux-ci sont ensuite normalisés pour obtenir des pourcentages. Si `graph=TRUE` alors un barplot est tracé pour afficher de manière plus graphique l'importance des variables.

2.2.9 Export du modèle : export_pmml(file_path="model.pmml", target_name)

Exporte le modèle au format PMML pour l'utiliser dans des outils externes.
`file_path` correspond au chemin pour sauvegarder le fichier.
`target_name` correspond au nom de la variable cible. Ce fichier enregistre les coefficients associés à chaque variable et ce pour chaque classe.

3 Détail des Calculs

Dans cette section, nous allons nous pencher sur le détail des calculs qui ont été effectués pour réaliser la régression logistique.
Tout d'abord, voici un pseudo code permettant d'expliquer la démarche. Cette démarche est tirée de `https://eric.univ-lyon2.fr/ricco/cours/slides/gradient_descent.pdf` :

- Identifier les classes uniques dans y
- Initialiser θ à zéro pour chaque classe
- Pour chaque classe k :
  - Créer y_k comme vecteur binaire pour la classe k
  - Pour chaque itération :
    - Calculer $h = \text{sigmoïde}(X * theta)$
    - Calculer le gradient de la fonction de coût
    - Ajouter les termes de régularisation au gradient si nécessaire
    - Mettre à jour θ
  - Stocker θ_k optimisé
- Combiner les paramètres optimisés pour chaque classe en une matrice
- Retourner la matrice des paramètres optimisés

Pour expliciter un peu plus la manière dont le calcul a été effectué, il faut savoir que nous avons utilisé la fonction de coût logistique (ou entropie croisée binaire) :

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log(h_\theta(x^{(i)})) + (1-y^{(i)})\log(1-h_\theta(x^{(i)}))\right]$$

Pour optimiser cette fonction de cout, il faut la dériver :

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)x_j^{(i)}$$

On peut la réécrire de manière matricielle :

$$\frac{\partial J(\theta)}{\partial \theta} = \frac{1}{m}X^T(h-y)$$

Cette dernière expression est le gradient que l'on a écrit dans le code.

4 Comparaison des performances

Dans cette étude, nous comparons les performances de deux implémentations de la régression logistique. La première est réalisée avec un package personnalisé en `R` et la deuxième utilise `scikit-learn` en `Python`. Nous évaluons la qualité des prédictions et la rapidité des calculs pour le dataset Iris.

4.1 Code et résultats sous R

Nous commençons par charger et préparer les données en `R`, puis nous entraînons le modèle de régression logistique à l'aide du package personnalisé.

4.1.1 Préparation des données

Le code suivant montre comment nous avons séparé les données en ensembles d'entraînement et de test :

```
# Exemple d'utilisation
# Charger les biblioth ques n cessaires
library(readr)

# Lire les fichiers CSV
train_data <- read_csv("train_data.csv")
test_data <- read_csv("test_data.csv")
```

```{r}
# S parer les caract ristiques (features) et la cible (target)
X_train <- train_data[, !(names(train_data) %in% c("target"))]
y_train <- train_data$target

X_test <- test_data[, !(names(test_data) %in% c("target"))]
y_test <- test_data$target
```

4.1.2 Entraînement du modèle

Le modèle est ensuite entraîné à l'aide de la régression logistique avec les paramètres spécifiés :

```
library(LogisticRegression)
# Entra nement du mod le
model <- LogisticRegression$new(penalty = NULL, lambda = 0,
                                l1_ratio = 0.5)
model <- model$fit(X_train, y_train)
#model$summary()
#model$export_pmml(target_name = "Species")
```

```
8
9  # Pr dictions sur l'ensemble de test
10 y_pred = model$predict(X_test)
11 result = model$test(y_test, y_pred, confusion_matrix = TRUE)
12 print(result)
```

4.1.3 Résultats sous R

Les résultats obtenus pour l'accuracy, la précision, le rappel et le F1-score sont les suivants :

Confusion Matrix :

	setosa	versicolor	virginica
setosa	18	0	0
versicolor	0	9	0
virginica	0	1	16

Accuracy : 0.9555
Precision : 0.9470
Recall : 0.9470
F1-Score : 0.9470

4.2 Code et résultats sous Python (scikit-learn)

Nous répétons une approche similaire en `Python`, utilisant `scikit-learn` pour entraîner la régression logistique et calculer les mêmes métriques.

4.2.1 Préparation des données

Voici le code Python pour charger et préparer les données Iris, ainsi que séparer l'ensemble d'entraînement et de test :

```
1  import numpy as np
2  import pandas as pd
3  from sklearn.model_selection import train_test_split
4  from sklearn.linear_model import LogisticRegression
5  from sklearn.metrics import accuracy_score, confusion_matrix
6  import matplotlib.pyplot as plt
7
8  # Charger les donn es iris depuis sklearn
9  from sklearn.datasets import load_iris
10 iris = load_iris()
11
12 # descriptives variables
13 X = pd.DataFrame(iris.data, columns=iris.feature_names)
```

```
# target
y = pd.DataFrame(iris.target, columns=['target'])

# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=123)


# Combine features and target for train and test datasets
train_data = pd.concat([X_train.reset_index(drop=True), y_train.reset_index(drop=True)], axis=1)
test_data = pd.concat([X_test.reset_index(drop=True), y_test.reset_index(drop=True)], axis=1)

# Save to CSV files to use them in our R package
train_data.to_csv('train_data.csv', index=False)
test_data.to_csv('test_data.csv', index=False)



# Initialisation et entra nement du mod le de r gression logistique
model = LogisticRegression(max_iter=200)  # max_iter est mis    200 pour s'assurer de la convergence
model.fit(X_train, y_train)

# Pr dictions sur l'ensemble de test
y_pred = model.predict(X_test)

# Affichage de la matrice de confusion
print("\nConfusion Matrix:")
print(confusion_matrix(y_test, y_pred))

# Calcul de la pr cision , du rappel et du F1-score moyens

# Matrice de confusion
cm = confusion_matrix(y_test, y_pred)

# Calcul de l'accuracy
accuracy = accuracy_score(y_test, y_pred)
# Calcul de la pr cision (precision) par classe
precision = np.diag(cm) / np.sum(cm, axis=0)
# Calcul du rappel (recall) par classe
recall = np.diag(cm) / np.sum(cm, axis=1)
# Calcul du F1-score par classe
f1_score = 2 * (precision * recall) / (precision + recall)
```

```
55
56 # Calcul des moyennes (moyenne pond r e par support)
57 precision_avg = np.mean(precision)
58 recall_avg = np.mean(recall)
59 f1_score_avg = np.mean(f1_score)
60
61 # Affichage des r sultats
62 print(f"\nAccuracy: {accuracy:.4f}")
63 print(f"Precision (moyenne) : {precision_avg:.4f}")
64 print(f"Recall (moyenne) : {recall_avg:.4f}")
65 print(f"F1-Score (moyenne) : {f1_score_avg:.4f}")
```

4.2.2 Résultats sous Python

Les résultats obtenus pour l'accuracy, la précision, le rappel et le F1-score sous Python sont les suivants :

Confusion Matrix :

	setosa	versicolor	virginica
setosa	18	0	0
versicolor	0	10	0
virginica	0	3	14

Accuracy : 0.9333
Precision : 0.9231
Recall : 0.9412
F1-Score : 0.9243

4.3 Comparaison des résultats

La comparaison des performances entre les deux modèles de notre package obtient des résultats supérieurs à ceux du modèle de sickit-learnR :

Métrique	R (Notre package)	Python (scikit-learn)
Accuracy	0.9555	0.9111
Precision	0.9470	0.9155
Recall	0.9470	0.9111
F1-Score	0.9470	0.9111

Les différences peuvent être dues à des variations dans les paramètres du modèle ou dans les implémentations spécifiques. Notre package semble offrir de meilleures performances en termes de prédiction.

4.4 Conclusion

Les modèles de régression logistique en `R` et `Python` ont montré des performances différentes sur le dataset Iris. Bien que les deux approches soient valides, `scikit-learn` semble offrir des résultats légèrement meilleurs en termes de précision et de rappel. Pour des applications réelles, il serait utile d'effectuer une validation croisée sur chaque modèle afin de mieux comprendre leurs capacités de généralisation.

5 Conclusion

Dans ce projet, nous avons développé un package R de régression logistique multinomiale permettant de réaliser des classifications supervisées sur des jeux de données avec des variables explicatives mixtes. Ce travail a impliqué une compréhension des algorithmes de descente de gradient et leur implémentation pour estimer les coefficients optimaux d'un modèle de régression logistique. L'utilisation de l'orientation objet avec des classes R6 a permis de structurer le code et de faciliter la construction de l'outil.

Nous avons comparé les performances de notre package avec celles d'outils existants tels que `scikit-learn` en Python, en utilisant des critères tels que la précision, le rappel, l'accuracy et le score F1. Les résultats obtenus montrent que notre implémentation est compétitive. La prise en charge des variables mixtes, la régularisation et l'export des modèles PMML ouvrent des perspectives intéressantes pour son utilisation dans des applications réelles.

De plus, contrairement à la fonction glm sur R, ce package permet d'étendre les régressions logistiques à des cas non binaires, offrant ainsi une plus grande flexibilité pour analyser des problèmes de classification multiclasses et d'autres types de régression complexes

Les prochains développements pourront se concentrer sur l'amélioration de la gestion de grands ensembles de données et l'optimisation des algorithmes d'entraînement pour des performances encore accrues. Cela peut passer entre autre par l'utilisation de librairies comme PySpark ou via une parallèlisation des calculs, notamment dans la fonction `reg_multinomiale()`, lorsque l'on itère sur les différentes classes le calcul de la descente de gradient.

Par ailleurs, des pistes à explorer incluent l'extension des options d'encodage des variables catégorielles qui aujourd'hui ne sont traitées que par un encodage 0-1 mais qui pourraient être traitées à l'aide d'une ACM.
De la même manière, créer la possibilité de choisir la fonction de perte semble être une piste intéressante afin de fournir une palette de choix à l'utilisateur plus importante.