Algorithmes Évolutionnaires — Master 2 MIAGE IA^2
Travaux dirigés N^o 5 : régression symbolique

Andrea G. B. Tettamanzi
Université côte d'Azur
andrea.tettamanzi@univ-cotedazur.fr

Année universitaire 2023/2024

Résumé

Dans cette séance, nous appliquerons la programmation génétique à un problème de régression symbolique. Pour ce faire, nous utiliserons le *framework* DEAP.

1 Introduction

Dans la régression, on cherche à ajuster les paramètres d'un modèle (linéaire, logistique, ou autre) pour approcher une variable à partir d'autres qui lui sont corrélées. Pourtant, la forme générale du modèle est donnée ou, dit autrement, le modèle est une hypothèse. La régression symbolique, par contre, consiste à chercher un modèle (c-à-d une fonction) qui approche la variable cible à partir des autres, sans faire d'hypothèses sur sa forme. La programmation génétique semble donc un outil parfaitement adapté pour ce genre d'exercice.

Nous allons utiliser le *framework* DEAP pour développer une approche de la régression symbolique basée sur la programmation génétique. Nous supposerons que le jeu de données du problème à résoudre soit donné dans un fichier en format TSV (de l'anglais *tab-separated values*), dont la première ligne contient les noms des variables et les lignes suivantes contiennent les observations dont ont dispose. La variable à approcher à partir des autres est, par convention, la dernière. Un exemple d'un tel fichier est le suivant :

```
x1   x2   y
1.0  0.0  1.0
0.0  1.0  1.0
0.0  0.0  0.0
1.0  1.0  2.0
1.0  2.0  3.0
2.0  1.0  3.0
```

2 Consignes

1. Vous pouvez vous inspirer du tutoriel avancé de DEAP sur la programmation génétique.
2. Choisissez un ensemble d'opérations primitives suffisant pour exprimer même les fonctions les plus complexes ; outre aux opérations arithmétiques, pensez à inclure aussi l'exponentiel, le logarithme et des fonctions trigonométriques.
3. Faites attention aux opérations/fonctions dont le domaine de définition ne coincide pas avec $\mathbb{R}$ (exemples : la racine carrée, le logarithme, la division) : pour éviter tout problème, redéfinissez-les pour les « protéger » des arguments hors domaine (comme le zéro au dénominateur pour la division, les nombres négatifs pour la racine carrée et les nombres non-positifs pour le logarithme).

4. Utilisez l'erreur quadratique moyenne comme critère d'optimisation (à minimiser). Soit y la variable cible, $x_1, \ldots, x_n$ les autres variables et f la fonction réalisée par l'individu à évaluer ; l'erreur quadratique moyenne de f est définie comme

$$\overline{EQ}(f) = \frac{1}{N} \sum_{i=1}^{N} (y^{(i)} - f(x_1^{(i)}, \ldots, x_n^{(i)}))^2, \tag{1}$$

   où N est le nombre d'observations fournies.

5. Créez un jeu de données bateau pour tester votre algorithme pendant son développement.
6. Une fois que vous êtes sûrs que votre algorithme marche correctement, essayez-le sur un jeu de données plus difficile. Vous pouvez en trouver en grand nombre, par exemple, sur le site `https://www.kaggle.com`. Veillez à en choisir un où les variables sont toutes numériques.
7. Vous constaterez peut-être que les fonctions trouvées par votre algorithme sont trop compliquées. Dans ce cas, vous pourriez décider de pénaliser les modèles inutilement compliqués en introduisant un autre critère d'optimisation, basé par exemple sur la taille de l'arbre syntaxique d'un individu. Vous pouvez le garder comme critère séparé (auquel cas, votre problème deviendra multi-objectif et vous chercherez donc un front de Pareto de solutions non dominées) ou bien le combiner avec l'erreur quadratique moyenne, en le traitant à l'instar d'une fonction de pénalisation).

Rendez votre code et vos observations par courriel, dans un archive zippé.