RAPPORT PDF
ML I supervised learning : project

Nicolas CHEN
Gael BARON
Sean MORTON

PARTIE 1 : Artificial Dataset Generation

Étapes du Code

Initialisation et Reproductibilité :

np.random.seed(42) : Cette ligne sert à initialiser le générateur de nombres aléatoires de NumPy avec une graine fixe (42). Cela garantit que le jeu de données généré est le même à chaque exécution du script, ce qui est crucial pour la reproductibilité des résultats.

Génération des Données de Base :

np.random.normal(loc=0, scale=1, size=(n_points, 6)) : Ici, nous créons un tableau de 300 points (lignes) et 6 colonnes, avec des valeurs tirées d'une distribution normale (gaussienne) de moyenne (loc) 0 et d'écart-type (scale) 1.

Ajustement des Moyennes et des Écarts Types :

Les moyennes et écarts types désirés pour chaque colonne sont définis dans les listes means et std_devs. Ensuite, pour chaque colonne, les valeurs sont ajustées pour correspondre à ces moyennes et écarts types spécifiques, en multipliant chaque valeur par l'écart type souhaité et en ajoutant la moyenne voulue.

Assurer les Exigences de Type de Données :

La première colonne est arrondie pour contenir uniquement des entiers, satisfaisant ainsi l'exigence d'avoir au moins une colonne d’entiers.

Ajout de Corrélations :

- **Positivement Corrélée** : Une nouvelle colonne est générée en multipliant les valeurs d'une colonne existante par un facteur (ici 1.2) ce qui crée une corrélation positive.
- **Négativement Corrélée** : De même, une corrélation négative est introduite en inversant le signe des valeurs d'une autre colonne, multiplié par un facteur (0.8).
- **Corrélation Proche de Zéro** : Une colonne indépendante des autres est générée en utilisant une nouvelle série de valeurs tirées d'une distribution normale, créant ainsi une corrélation proche de zéro avec les autres colonnes.

Sauvegarde en CSV :

Le DataFrame est créé avec pd.DataFrame en utilisant les données générées et est sauvegardé dans un fichier CSV, artificial_dataset.csv. Chaque colonne est nommée séquentiellement (Colonne1, Colonne2, etc.).

Description des Colonnes:

1. Colonne1 : Contient des entiers. Les valeurs sont initialement générées à partir d'une distribution normale, puis ajustées pour avoir une moyenne et un écart type spécifiques, et enfin arrondies pour devenir des entiers.

2. Colonne2 : Les valeurs sont ajustées pour avoir une moyenne et un écart type spécifiques, et servent de base pour la colonne positivement corrélée ajoutée plus tard.

3. Colonne3 : Comme pour la Colonne2, mais utilisée pour générer une colonne négativement corrélée.

4. Colonne4 à Colonne6 : Chacune ajustée pour avoir sa propre moyenne et écart type uniques, contribuant à la diversité du jeu de données.

7. Colonne ajoutée pour corrélation positive : Générée à partir de la Colonne2, elle est conçue pour avoir une corrélation positive avec cette colonne.

8. Colonne ajoutée pour corrélation négative : Créée en inversant et en ajustant les valeurs de la Colonne3, introduisant une corrélation négative.

9. Colonne pour corrélation proche de zéro : Générée indépendamment des autres pour assurer une corrélation minimale avec le reste du jeu de données.

PARTIE 2: Definition of a Metric

Ce programme a pour objectif de calculer une matrice de dissimilarité entre les échantillons d'un jeu de données en tenant compte à la fois des caractéristiques numériques et catégorielles.

L'utilisation de **from scipy.spatial.distance import euclidean** dans le programme permet d'accéder directement à la fonction **euclidean** pour calculer la distance euclidienne entre deux points ou, dans le contexte de notre programme, entre deux échantillons du jeu de données basés sur leurs caractéristiques numériques.

Chargement du jeu de données

Le jeu de données est chargé à partir d'un fichier CSV. Ce jeu de données est supposé contenir à la fois des caractéristiques numériques (entiers ou flottants) et catégorielles (objets ou chaînes de caractères).

Normalisation des caractéristiques numériques

Les caractéristiques numériques sont normalisées pour qu'elles soient toutes sur une échelle de 0 à 1. Cette normalisation est réalisée en soustrayant la valeur minimale de chaque caractéristique et en divisant par son amplitude (valeur maximale moins valeur minimale). Cela permet de s'assurer que les différentes échelles des caractéristiques numériques n'affectent pas la mesure de dissimilarité.

Calcul de la dissimilarité

La dissimilarité entre deux échantillons est calculée en combinant la dissimilarité des caractéristiques numériques et catégorielles.
- Pour les caractéristiques numériques, la dissimilarité est mesurée par la distance euclidienne entre les échantillons, après normalisation.
- Pour les caractéristiques catégorielles, une mesure simple est utilisée : la dissimilarité est de 0 si les deux échantillons partagent la même catégorie, et de 1 autrement.

Ces dissimilarités sont ensuite combinées pour obtenir une mesure de dissimilarité totale entre chaque paire d'échantillons.

Matrice de dissimilarité

Une matrice carrée est construite pour enregistrer la dissimilarité entre tous les paires d'échantillons dans le jeu de données. Chaque élément `(i, j)` de cette matrice représente la dissimilarité entre l'échantillon `i` et l'échantillon `j`.

Pour optimiser le calcul, la symétrie de la matrice est exploitée : puisque la dissimilarité entre `i` et `j` est la même que celle entre `j` et `i`, seule la moitié de la matrice est calculée explicitement.

Statistiques de la dissimilarité

Après le calcul de la matrice de dissimilarité, le programme calcule la moyenne et l'écart type de toutes les valeurs de dissimilarité. Cela donne une idée de la

dissimilarité générale au sein du jeu de données et de la variabilité de cette dissimilarité.

Sauvegarde de la matrice de dissimilarité

Enfin, la matrice de dissimilarité est sauvegardée dans un fichier au format `.npy`, permettant son utilisation ultérieure pour des analyses ou des visualisations.

PARTIE 3 : Prediction of the winner of a NBA game (classification)

Étapes du Programme

- **Chargement des données**: Les datasets X_train, y_train, X_test, y_test sont chargés à partir des fichiers .npy. Ces datasets contiennent respectivement les caractéristiques d'entrainement, les étiquettes d'entrainement, les caractéristiques de test, et les étiquettes de test.

- **Normalisation des données**: Les données sont normalisées en utilisant StandardScaler pour s'assurer que toutes les caractéristiques sont sur une échelle similaire. Cette étape est cruciale pour de nombreux algorithmes de machine learning afin de garantir que les caractéristiques avec des ordres de grandeur plus importants ne dominent pas ceux avec des ordres de grandeur plus petits.

- **Initialisation d'un dictionnaire pour stocker les scores de précision**: Un dictionnaire accuracy_scores est créé pour enregistrer la précision de chaque modèle testé.

- **Fonction pour entraîner et évaluer un modèle**: La fonction train_evaluate_model prend un modèle et son nom comme arguments. Elle entraîne le modèle sur le dataset d'entraînement (après normalisation), fait des prédictions sur le dataset de test, calcule la précision des prédictions par rapport aux vraies étiquettes de test et stocke la précision dans le dictionnaire accuracy_scores.

Entraînement et évaluation des modèles:

- **Régression Logistique**: Un modèle linéaire utilisé pour les problèmes de classification.

- **SVC (Support Vector Classifier)**: Un modèle basé sur les machines à vecteurs de support, utile pour des frontières de décision complexes.

- **KNN (K-Nearest Neighbors)**: Un modèle basé sur la proximité des voisins pour la classification.

- **MLP (Multi-layer Perceptron classifier)**: Un type de réseau de neurones pour la classification.
- **AdaBoost**: Un algorithme d'ensemble qui combine plusieurs modèles faibles pour créer un modèle fort.

Résultats

- **Comparaison des Performances**: On s'attend à ce que les modèles présentent des performances variables en raison de la nature des données et des hypothèses inhérentes à chaque algorithme. Par exemple, les modèles linéaires comme la régression logistique pourraient ne pas performer aussi bien si les données ne sont pas linéairement séparables. D'autre part, des modèles plus complexes comme MLP ou AdaBoost pourraient capturer des relations plus complexes mais au risque de surajuster, surtout si les données d'entraînement sont limitées ou si le modèle est trop complexe.

- **Impact des Hyperparamètres**: Les performances de ces modèles sont également fortement influencées par leurs hyperparamètres. Dans ce programme, des valeurs par défaut sont utilisées pour la plupart des modèles, à l'exception de max_iter pour la régression logistique et MLP, qui est augmenté pour garantir la convergence. Un ajustement fin des hyperparamètres (par exemple, via une recherche en grille comme GridSearchCV pour certains modèles) pourrait améliorer significativement les performances.

- **Choix du Meilleur Modèle**: Le modèle avec la plus haute précision est identifié comme le meilleur. Cependant, il est important de noter que la précision seule ne fournit pas une image complète de la performance d'un modèle. D'autres métriques comme le rappel, la précision (dans le sens de la métrique, pas le modèle), le score F1, ou l'analyse de la courbe ROC pourraient fournir des insights supplémentaires, surtout dans le cas de déséquilibre de classe.

- **Conclusion**: La sélection du "meilleur" modèle dépendra non seulement de la précision obtenue sur le jeu de test mais aussi de considérations supplémentaires telles que la complexité du modèle, le temps d'entraînement, et la capacité à généraliser à de nouvelles données non vues.

PARTIE 4 : Prediction of the amount of electricity produced (regression)

Ce script Python est destiné à l'évaluation et à l'optimisation de deux modèles de régression, Lasso et MLPRegressor, pour prédire une cible numérique à partir d'un ensemble de caractéristiques. Il utilise des ensembles de données divisés en entraînement et en test pour entraîner les modèles et évaluer leurs performances. Voici une explication détaillée de chaque étape et des résultats obtenus :

Chargement des ensembles de données

Les ensembles de données pour l'entraînement (X_train, y_train) et le test (X_test, y_test) sont chargés à partir de fichiers .npy. Les vecteurs cibles (y_train et y_test) sont convertis en 1D pour s'assurer qu'ils sont dans le format approprié pour l'entraînement et l'évaluation des modèles.

Optimisation du modèle Lasso

- **Création d'un pipeline** : Un pipeline comprenant une étape de normalisation (StandardScaler) et une étape de régression Lasso est créé. La normalisation est une étape cruciale pour de nombreux algorithmes de machine learning pour s'assurer que toutes les caractéristiques sont sur une échelle comparable.
- **Recherche des meilleurs hyperparamètres** : GridSearchCV est utilisé pour trouver la meilleure valeur de l'hyperparamètre alpha pour le modèle Lasso. L'hyperparamètre alpha contrôle la force de la régularisation Lasso, qui peut aider à prévenir le surajustement en introduisant des pénalités sur la taille des coefficients. Différentes valeurs de alpha sont testées, et la performance est évaluée à l'aide de la validation croisée avec 5 sous-ensembles.
- **Évaluation sur l'ensemble de test** : Le meilleur modèle Lasso est évalué sur l'ensemble de test, et le score R^2 est calculé pour mesurer la performance du modèle. Le score R^2 indique la proportion de la variance de la variable dépendante qui est prévisible à partir des variables indépendantes, avec des valeurs plus proches de 1 indiquant une meilleure performance.

Optimisation du MLPRegressor

- **Création d'un pipeline** : Un pipeline similaire est créé pour le MLPRegressor, incluant également une étape de normalisation suivie par le régresseur de perceptron multicouche.
- **Recherche des meilleurs hyperparamètres** : La recherche des meilleurs hyperparamètres pour le MLPRegressor inclut la taille des couches cachées (hidden_layer_sizes) et le terme de régularisation alpha. Comme pour Lasso, GridSearchCV est utilisé avec une validation croisée pour identifier la meilleure combinaison d'hyperparamètres.

- **Évaluation sur l'ensemble de test** : Le MLPRegressor optimisé est ensuite évalué sur l'ensemble de test, et le score R^2 est utilisé pour évaluer sa performance.

Résultats

Le script affiche les meilleurs paramètres trouvés pour chaque modèle ainsi que leurs scores R^2 sur l'ensemble de test. Enfin, il compare les performances des deux modèles pour identifier le plus efficace, c'est-à-dire celui ayant le score R^2 le plus élevé sur l'ensemble de test, et affiche cette méthode ainsi que son score.

En résumé, ce script effectue une analyse comparative de deux modèles de régression en utilisant une recherche exhaustive des hyperparamètres et évalue leur capacité à prédire précisément les données de test à l'aide du score R^2. La méthode affichant le meilleur score R^2 est considérée comme la plus performante pour cet ensemble de données spécifique.