Papport de Projet

Project Name: LinearRegression - Prédictions de Prix de Loyer

Team Members:

Asmae ELhakioui

KHALD Adam

Yassine Mokhass

Mohamed Amine Darraj

HINIMDOU MORSIA GUITDAM

Team Leader: HINIMDOU MORSIA GUITDAM

Date: Janvier 2025

Contents

1 Model Description

1.1 Algorithm in Pseudo-Code

Paramètres et données

1. **Initialisation** :
   - `seed = 42`
   - `n_lignes = 10 000`
2. **Bornes des variables** :
   - `Surface` : $30 \rightarrow 300$
   - `Âge` : $0 \rightarrow 49$
   - `Sécurité`, `Localisation`, `Chambres`, `Équipement` : $1 \rightarrow 5$
3. **Génération des données** :
   - `Surface` : uniforme `[min_surface, max_surface]`
   - `Âge` : entiers aléatoires `[min_age, max_age]`
   - Autres variables : entiers aléatoires dans leurs bornes respectives.
4. **Calcul du prix** :

$$\begin{aligned}\text{Prix} =& (2000 \cdot \text{Surface}) - (500 \cdot \text{Âge}) + (1500 \cdot \text{Sécurité}) \\ &+ (3000 \cdot \text{Localisation}) + (10000 \cdot \text{Chambres}) \\ &+ (2000 \cdot \text{Équipement}) + \text{bruit gaussien}\end{aligned}$$

5. **Ajout d'outliers** :
   - `pourcentage_outliers = 0.02` $\rightarrow$ `n_outliers = round(pourcentage_outliers` $\cdot$ `n_lignes)`
   - Modifier `Surface`, `Âge`, et `Prix` avec des facteurs aléatoires.

Préparation des données

1. **Construction du DataFrame** : inclure toutes les variables et `Prix`.
2. **Matrice des caractéristiques** :
   - `X` : normalisation (sans biais), ajouter une colonne biais (1).
   - `y` : vecteur des prix.
3. **Division des ensembles** :
   - `80%` train, `20%` test $\rightarrow$ `X_train, y_train, X_test, y_test`.

Descente de gradient avec mini-lots

1. **Paramètres** :
   - `learning_rate=0.001`, `epochs=10 000`, `batch_size=128`, `tolerance=1e-6`.
2. **Étapes** :
   - Initialiser `theta = 0`.
   - Pour chaque époque :
     - Mélanger les données.
     - Pour chaque mini-lot :
       * Calculer prédictions, erreurs, gradients.
       * Mettre à jour `theta`.
     - Calculer MSE, vérifier convergence (arrêt si tolérance atteinte).
   - Retourner `theta`.

Modèle avec équation normale

- Calculer `theta_normal =` $(X^T X)^{-1} X^T y$.

Prédictions et évaluation

1. **Prédictions** :
   - `y_pred_gd` avec `theta_gd`.
   - `y_pred_normal` avec `theta_normal`.
2. **Metrics** :
   - **MSE** : erreur quadratique moyenne.
   - **R²** : $1 - \frac{\text{MSE}}{\text{variance de } y}$
   - **Précision** : $1 - \frac{\text{MSE}}{\text{variance}}$
3. **Affichage** :
   - Afficher `MSE`, `R²`, `précision` pour chaque méthode.
   - Afficher `theta_gd` et `theta_normal`.

2 Simulation Results

2.1 Programming Language 1: [Python]

Équation Normale

L'approche par équation normale constitue notre première méthode d'analyse. Les résultats obtenus sont :

$$\text{MSE} = 6\,727\,500\,090,08$$
$$R^2 = 0,8837$$

Cette approche démontre une capacité satisfaisante à expliquer la variance des prix, avec près de 88,37% de la variance totale expliquée par le modèle.

Descente de gradient par mini-lots (version simple)

La seconde approche implémentée utilise une descente de gradient stochastique par mini-lots dans sa forme basique. Les métriques obtenues sont :

$$\begin{aligned} \text{MSE} &= 17\,335\,629\,042,27 \\ \text{MAE} &= 114\,996,68 \\ \text{RMSE} &= 131\,664,84 \\ R^2 &= 0,2891 \end{aligned}$$

Cette implémentation montre des limitations significatives :

- Un faible coefficient de détermination ($R^2 = 0,2891$)
- Une erreur quadratique moyenne élevée
- Une tendance à la sous-estimation systématique pour les biens de grande surface

Descente de gradient par mini-lots avec régularisation et standardisation

La version améliorée incorpore deux modifications majeures :

- Standardisation des variables d'entrée
- Introduction d'un terme de régularisation

Paramètres explorés

Une recherche par grille a été effectuée sur les hyperparamètres suivants :

Paramètre	**Valeurs testées**
Taux d'apprentissage (Learning rate)	{0,0001; 0,001; 0,01}
Taille des lots (Batch size)	{16; 32; 64}
Coefficient de régularisation (λ)	{0; 0,01; 0,1; 1}

Configuration optimale

Les meilleurs hyperparamètres identifiés sont :

- Taux d'apprentissage : 0,01
- Taille des lots : 64
- Coefficient de régularisation (λ) : 0,01

Performances du modèle optimal

Le modèle optimisé atteint des performances remarquables :

$$\begin{aligned} \text{MSE} &= 9\,922\,421,87 \\ \text{MAE} &= 7\,953,77 \\ \text{RMSE} &= 9\,961,14 \\ R^2 &= 0,9959 \end{aligned}$$

Analyse comparative des performances

L'évolution des performances à travers les différentes approches montre une amélioration significative :

Métrique	**Mini-batch simple**	**Équation normale**	**Mini-batch optimisé**
R^2	0,2891	0,8837	0,9959
MSE	17,34 $\times 10^9$	6,73 $\times 10^9$	9,92 $\times 10^6$

2.2 Programming Language 2: [Julia]

Voici les résultats obtenus pour les différentes métriques de régression sur le jeu de données X_{train} de dimension 7.

Métriques de performance

- **Erreur quadratique moyenne (MSE)** avec la descente de gradient (mini-batch) : $1.9547565932288487 \times 10^9$
- **MSE** avec l'équation normale : $1.954749761726393 \times 10^9$
- **Coefficient de détermination** R^2 avec la descente de gradient (mini-batch) : 0.9244379322245169
- **$\mathbf{R^2}$** avec l'équation normale : 0.9244381962995701
- **Précision** avec la descente de gradient (mini-batch) : 0.9244757132584046
- **Précision** avec l'équation normale : 0.9244759772014204

Coefficients obtenus

Les coefficients des variables prédictives obtenus par la descente de gradient et par l'équation normale sont les suivants :

- **Coefficients obtenus par la descente de gradient :**

$$\hat{\beta} = [367863.03, 155927.10, -6667.00, 2096.87, 4717.80, 14180.68, 4239.29]$$

- **Coefficients obtenus par l'équation normale :**

$$\hat{\beta} = [367862.14, 155907.65, -6656.21, 2101.47, 4706.50, 14177.92, 4236.33]$$

2.3 Programming Language 3: [R]

Métriques d'évaluation: Les résultats obtenus montrent des performances similaires entre les deux méthodes. Pour la **descente de gradient avec mini-lots**, l'erreur quadratique moyenne (MSE) est de 4180382579, tandis que la méthode de **l'équation normale** donne une MSE de 4180870660.

Les coefficients de détermination (R^2) sont également très proches : 0,8994 pour la descente de gradient et 0,8994 pour l'équation normale. La précision suit la même tendance avec respectivement 0,8994 et 0,8994 pour les deux méthodes.

Coefficients estimés: Les coefficients estimés par les deux méthodes sont présentés ci-dessous :

Variable	Descente de gradient	Équation normale
Constante	384324,714	384324,852
Surface	209652,696	209607,429
Âge	-7106,182	-7087,802
Sécurité	2300,411	2285,825
Localisation	5696,214	5695,258
Chambres	13533,690	13535,694
Équipement	3754,511	3758,108

Table 1: Comparaison des coefficients estimés par les deux méthodes

3 Comparative Analysis of Results

Méthode de l'Équation Normale

Les résultats obtenus par l'approche de l'équation normale sont similaires dans les trois langages utilisés :

- **Python :** $R^2 = 0.8837$, MSE = 6,727,500,090.08
- **Julia :** $R^2 = 0.9244$, MSE = $1.9547 \times 10\hat{9}$
- **R :** $R^2 = 0.8994$, MSE = 4,180,382,579

Ces résultats montrent que l'équation normale offre une bonne explication de la variance des données, avec un R^2 élevé dans tous les cas.

Descente de Gradient par Mini-lots (Version Simple)

L'approche de descente de gradient par mini-lots dans sa version simple montre des performances nettement inférieures en Python, avec un R^2 de seulement 0.2891, tandis que Julia et R montrent des résultats beaucoup plus performants, avec un R^2 proche de 0.92.

- **Python (Simple) :** $R^2 = 0.2891$, MSE = 17,335,629,042.27

- **Julia :** $R^2 = 0.9244$, MSE = $1.9548 \times 10\hat{9}$
- **R :** $R^2 = 0.8994$, MSE = 4,180,382,579

Cela indique que la version simple de la descente de gradient par mini-lots nécessite des améliorations dans sa mise en œuvre.

Optimisation par Standardisation et Régularisation

L'approche optimisée en Python, avec la standardisation des données et l'introduction de régularisation, montre des performances exceptionnelles avec un R^2 de 0.9959 et un MSE fortement réduit :

- **Python (Optimisé) :** $R^2 = 0.9959$, MSE = 9,922,421.87

Cela montre que l'optimisation par standardisation et régularisation améliore significativement les performances des modèles, comparativement aux versions non optimisées.

Comparaison des Coefficients Estimés

Les coefficients estimés par la descente de gradient et par l'équation normale sont très similaires entre Python, Julia, et R. Cependant, de petites différences peuvent être observées en raison des particularités des algorithmes et des jeux de données.

- **Python (Mini-batch) :** $\beta = [367, 863.03, 155, 927.10, -6, 667.00, 2, 096.87, 4, 717.80, 14, 180.68, 4$
- **Julia (Mini-batch) :** $\beta = [367, 862.14, 155, 907.65, -6, 656.21, 2, 101.47, 4, 706.50, 14, 177.92, 4, 2$

Les coefficients estimés sont proches, mais quelques écarts existent entre les valeurs obtenues par les différentes méthodes.

Résumé des Performances

Méthode	R^2	MSE
Python (Simple Mini-batch)	0.2891	17,335,629,042.27
Python (Normal Equation)	0.8837	6,727,500,090.08
Python (Optimisé)	0.9959	9,922,421.87
Julia (Mini-batch)	0.9244	$1.9548 \times 10\hat{9}$
Julia (Normal Equation)	0.9244	$1.9547 \times 10\hat{9}$
R (Mini-batch)	0.8994	4,180,382,579
R (Normal Equation)	0.8994	4,180,870,660

Conclusion

L'optimisation par **standardisation** et **régularisation** en **Python** donne les meilleures performances globales, avec un R^2 proche de 1.0. Les approches en **Julia** et **R** fournissent des résultats solides, mais l'optimisation en Python reste supérieure. Cette analyse démontre l'importance de l'optimisation pour améliorer les performances des modèles de régression.

4 Complexity Analysis

1. Génération des données

- **Création des caractéristiques** : Chaque caractéristique (`surface`, `age`, etc.) est générée avec des boucles implicites sur n lignes.

  $$\text{Complexité} : O(n)$$

  pour chaque colonne, soit $O(kn) = O(n)$, où $k = 6$ (nombre de colonnes).

- **Ajout des outliers** :
  - Sélection aléatoire des indices : $O(n)$.
  - Modification des valeurs des k outliers : $O(k)$, où $k = 0.02n$.

  $$\text{Complexité totale} : O(n)$$

  car $k = 0.02n$ reste proportionnel à n.

2. Préparation des données

- **Construction de X et y** : Concaténation des colonnes dans une matrice X et extraction des prix dans y.

  $$\text{Complexité} : O(nm)$$

  où $m = 6$ (nombre de caractéristiques).

- **Normalisation avec `scale()`** : Parcours de chaque colonne de X pour calculer la moyenne et l'écart-type.

  $$\text{Complexité} : O(nm)$$

- **Division en ensembles training/testing** :
  - Sélection aléatoire : $O(n)$.
  - Extraction des données : $O(nm)$.

  $$\text{Complexité totale} : O(nm)$$

3. Descente de Gradient Mini-Batch

La descente de gradient effectue des itérations sur `epochs` époques, avec des mini-batchs de taille b.

- **Étapes principales** :
  - Permutation aléatoire des données : $O(n)$.
  - Calcul des gradients pour chaque batch (X_{batch} de taille b) :
    - Prédictions : $O(bm)$,

  - Calcul des erreurs : $O(b)$,
  - Gradients : $O(bm)$,
  - Mise à jour des paramètres : $O(m)$.

- **Coût par époque** :
$$O(nm + (n/b) \times bm) = O(nm)$$

- **Total sur toutes les époques** :
$$O(\text{epochs} \times nm)$$

4. Résolution par Équation Normale

- **Calcul de $X^T X$** : Multiplication matricielle ($m \times n$ par $n \times m$).
$$O(nm^2)$$

- **Inversion de $X^T X$** : Inversion d'une matrice $m \times m$.
$$O(m^3)$$

- **Calcul de $X^T y$** : Multiplication matricielle ($m \times n$ par $n \times 1$).
$$O(nm)$$

- **Multiplication finale** $(X^T X)^{-1} X^T y$:
$$O(m^2)$$

- **Complexité totale** :
$$O(nm^2 + m^3)$$

5. Prédictions et Évaluation

- **Prédictions ($X\theta$)** : Multiplication matricielle ($n \times m$ par $m \times 1$).
$$O(nm)$$

- **Calcul du MSE et R^2** : Calcul des erreurs au carré et de leur moyenne.
$$O(n)$$

Récapitulatif des Complexités

Étape	Descente de Gradient (O)	Équation Normale (O)
Génération des données	$O(n)$	$O(n)$
Préparation des données	$O(nm)$	$O(nm)$
Entraînement	$O(\text{epochs} \times nm)$	$O(nm^2 + m^3)$
Prédictions	$O(nm)$	$O(nm)$
Évaluation	$O(n)$	$O(n)$