Contents

RapportCodePython

mouhamadou lamine cisse

January 2023

1 Introduction

1.1 Présentation du challenge et de l'objectif du projet

Le challenge proposé consiste à développer un programme capable d'estimer l'année d'une chanson à partir de son contenu sonore. Pour cela, vous disposez d'un ensemble de données comprenant 4578 chansons avec leurs années associées et leurs descripteurs sonores, ainsi que 2289 chansons supplémentaires pour lesquelles les années ne vous sont pas fournies. Notre objectif est de développer un programme capable d'estimer les années de ces chansons avec la plus grande précision possible. Pour cela, nous allons utiliser des algorithmes d'apprentissage automatique pour entraîner notre programme sur l'ensemble de données d'apprentissage (les 4578 chansons avec leurs années connues) et évaluer sa performance sur l'ensemble de données de test (les 2289 chansons pour lesquelles les années ne sont pas connues). La précision de notre programme sera évaluée en comparant les années estimées à celles qui sont réellement associées aux chansons de l'ensemble de données de test. Plus le programme sera capable d'estimer les années de manière précise, plus sa performance sera élevée. En résumé, l'objectif de ce projet est de développer un programme capable de dater des chansons de manière automatique en utilisant des algorithmes d'apprentissage automatique et en évaluant la précision de sa performance sur un ensemble de données de test.

1.2 Description des données utilisées et de la méthodologie de programmation

Nous avons utilisé un jeu de données comprenant 4578 chansons avec leurs années de sortie et leurs descripteurs sonores, ainsi que 2289 chansons supplémentaires pour lesquelles les années ne sont pas connues. Ces données étaient stockées dans un fichier au format .npz et étaient organisées sous forme de matrices et de vecteurs. La matrice "X_labeled" contenait les descripteurs sonores de chaque chanson, tandis que le vecteur "y_labeled" contenait les années associées aux chansons. La matrice "X_unlabeled" contenait les descripteurs sonores des chansons pour lesquelles les années ne sont pas connues.
Pour mettre en place le modèle d'apprentissage automatique, nous avons utilisé l'ensemble de données "X_labeled" et "y_labeled" comme données d'apprentissage. Nous avons entraîné le modèle sur cet ensemble de données en utilisant un ou plusieurs algorithmes d'apprentissage automatique. Notre objectif était d'obtenir un modèle capable de prédire les années de sortie des chansons avec la plus grande précision possible.
Une fois que le modèle a été entraîné sur les données d'apprentissage, nous avons évalué ses performances sur l'ensemble de données de test "X_unlabeled" en utilisant une métrique adéquate. Cela nous a permis de mesurer la qualité du modèle et de vérifier s'il était capable de générer des prédictions précises pour des chansons dont les années ne sont pas connues.

1.3 Présentation des objectifs

Les objectifs de ce rapport sont de présenter en détail les différentes étapes de notre projet de développement d'un programme capable d'estimer l'année de sortie d'une chanson à partir de son contenu sonore. Nous allons expliquer les choix de la méthodologie de programmation que nous avons suivie et présenter les algorithmes d'apprentissage automatique que nous avons utilisés pour résoudre ce problème.

Nous présenterons également les résultats obtenus et analyserons les performances de chaque algorithme.

2 Lecture et manipulation des données

Pour lire les données contenues dans un fichier au format npz, nous avons utilisé la fonction load_data(filename) qui est data_utils.py. Cette fonction prend en argument une chaîne de caractères désignant le nom du fichier de données et renvoie trois tableaux de nombres à n dimensions (nd-arrays) : une matrice X_labeled, un vecteur y_labeled et une matrice X_unlabeled. Ces trois tableaux correspondent respectivement aux exemples étiquetés (avec leurs années connues), aux étiquettes associées à ces exemples (les années des chansons) et aux exemples non étiquetés (avec des années inconnues).
Avant d'entraîner nos modèles , nous avons utilisé la fonction randomize_data qui permute de façon simultanée l'ordre de X et Y, et la fonction split_data qui prend en argument une matrice X et un vecteur y et renvoie leurs versions mélangées aléatoirement. La figure suivante nous donne un aperçu sur les données d'apprentissage.

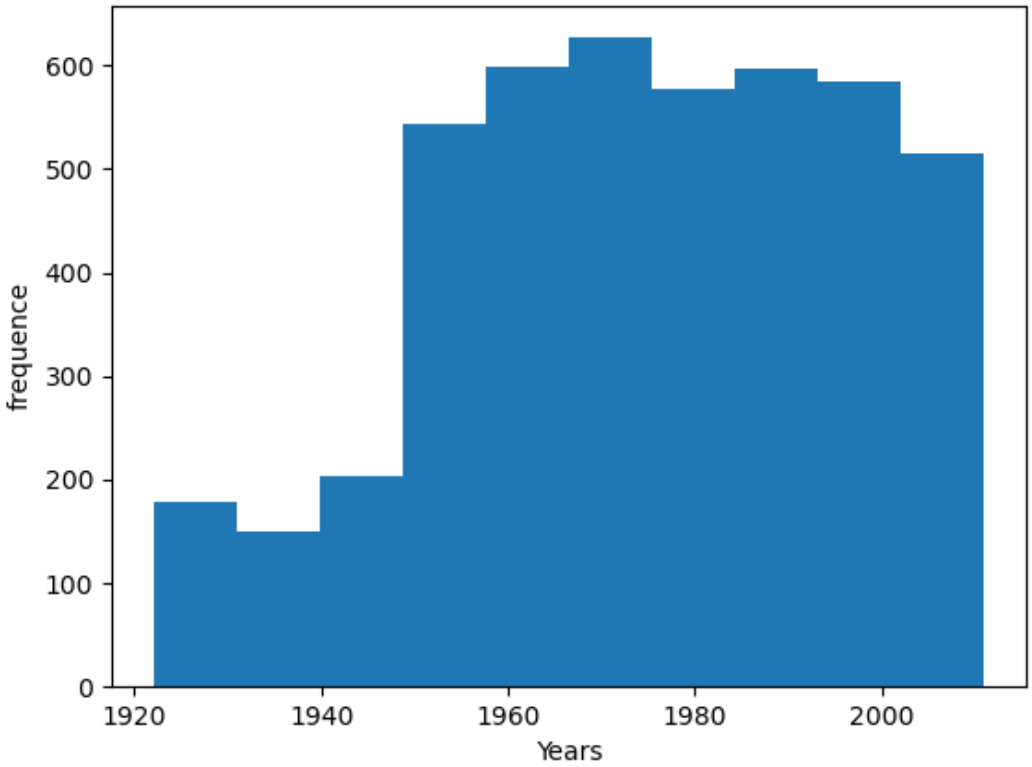

Figure 1: Repartition des etiquettes(age) en fonction de l'année.

3 Algorithmes d'apprentissage élémentaires

3.1 Régression linéaire

La régression linéaire est un algorithme d'apprentissage supervisé qui permet de modéliser une relation entre une variable cible (y) et un ou plusieurs variables explicatives (x). La régression linéaire consiste à trouver une fonction de prédiction de la forme y = f(x) = $X \cdot w + b$, où w et b sont des paramètres à estimer à partir d'un ensemble d'apprentissage (x, y). Le but est de trouver les valeurs

de w et b qui permettent de minimiser l'erreur quadratique moyenne entre les valeurs prédites et les valeurs réelles de y. nous avons utilisé la classe LinearRegression qui est dans le fichier linear_regression.py pour le coder.

3.2 Regression lineaire Least Square

La classe LinearRegressionLeastSquares hérite de la classe LinearRegression et implémente l'algorithme de régression linéaire aux moindres carrés. La méthode fit prend en argument un ensemble d'apprentissage (X, y) et utilise une méthode de moindres carrés pour estimer les paramètres w et b qui minimisent l'erreur quadratique moyenne entre les valeurs prédites et les valeurs réelles de y.
Pour cela, la méthode fit ajoute une colonne de 1 à la matrice d'exemples X et calcule sa pseudo-inverse en utilisant la fonction np.linalg.pinv de NumPy. Ensuite, la méthode calcule le produit matriciel entre la pseudo-inverse et y en utilisant la fonction np.dot de NumPy. Les coefficients w et b sont extraits du résultat du produit matriciel et stockés dans les attributs w et b de l'objet. La méthode predict de la classe LinearRegression pourra ensuite être utilisée pour prédire des étiquettes y à partir d'un ensemble d'exemples X en utilisant la fonction de prédiction y = wx + b avec les valeurs de w et b calculées par la méthode fit.

3.3 Regression lineaire mean

La classe LinearRegressionMean hérite de la classe LinearRegression et implémente une stratégie très simple pour l'apprentissage d'une fonction de prédiction linéaire. La fonction de prédiction est constante et égale à la moyenne des étiquettes d'apprentissage y.
La méthode fit de cette classe prend en argument un ensemble d'apprentissage (X, y) et calcule la moyenne des étiquettes y en utilisant la fonction np.mean de NumPy. Le coefficient w est initialisé à un vecteur de zéros de même taille que le nombre de caractéristiques dans X, tandis que le coefficient b est initialisé à la moyenne des étiquettes y. Les coefficients w et b sont stockés dans les attributs w et b de l'objet. La méthode predict de la classe LinearRegression pourra ensuite être utilisée pour prédire des étiquettes y à partir d'un ensemble d'exemples X en utilisant la fonction de prédiction y = wx + b avec les valeurs de w et b calculées par la méthode fit.
Cette stratégie est très simple et peu performante, mais elle peut être intéressante pour interpréter les résultats et comparer avec d'autres stratégies.

3.4 Regression lineaire median

La classe LinearRegressionMedian hérite de la classe LinearRegression et implémente une stratégie très simple pour l'apprentissage d'une fonction de prédiction linéaire. La fonction de prédiction est constante et égale à la médiane des étiquettes d'apprentissage y.
La méthode fit de cette classe prend en argument un ensemble d'apprentissage (X, y) et calcule la médiane des étiquettes y en utilisant la fonction np.median de NumPy. Le coefficient w est initialisé à un vecteur de zéros de même taille que le nombre de caractéristiques dans X, tandis que le coefficient b est initialisé à la médiane des étiquettes y. Les coefficients w et b sont stockés dans les attributs w et b de l'objet. La méthode predict de la classe LinearRegression pourra ensuite être utilisée pour prédire des étiquettes y à partir d'un ensemble d'exemples X en utilisant la fonction de prédiction y = wx + b avec les valeurs de w et b calculées par la méthode fit.
Cette stratégie est très simple et peu performante, mais elle peut être intéressante pour interpréter les résultats et comparer avec d'autres stratégies. La médiane peut être préférée à la moyenne dans certains cas, par exemple lorsque les étiquettes y ont des valeurs extrêmes (outliers) qui peuvent fausser la moyenne. La médiane est moins sensible aux valeurs extrêmes et peut donner une meilleure représentation de la distribution des étiquettes y.

3.5 Regression lineaire majoritaire

La classe LinearRegressionMajority hérite de la classe LinearRegression et implémente une stratégie très simple pour l'apprentissage d'une fonction de prédiction linéaire. La fonction de prédiction est constante et égale à l'étiquette d'apprentissage la plus fréquente (majorité).

La méthode fit de cette classe prend en argument un ensemble d'apprentissage (X, y) et calcule l'histogramme des étiquettes y en utilisant la fonction np.histogram de NumPy. Le coefficient w est initialisé à un vecteur de zéros de même taille que le nombre de caractéristiques dans X, tandis que le coefficient b est initialisé à l'étiquette la plus fréquente (c'est-à-dire celle qui apparaît le plus souvent dans y). Les coefficients w et b sont stockés dans les attributs w et b de l'objet. La méthode predict de la classe LinearRegression pourra ensuite être utilisée pour prédire des étiquettes y à partir d'un ensemble d'exemples X en utilisant la fonction de prédiction y = wx + b avec les valeurs de w et b calculées par la méthode fit.
Cette stratégie est très simple et peu performante, mais elle peut être intéressante pour interpréter les résultats et comparer avec d'autres stratégies. Elle peut être utilisée lorsque les étiquettes y sont distribuées de manière assez uniforme et que la majorité des étiquettes a une valeur distincte de la minorité.

4 Expérience avec ressources limitées

Dans cette partie, on va mesurer les performances de différents algorithmes d'apprentissage sur un ensemble de données donné. On va tout d'abord charger les données depuis le fichier YearPrediction-MSD_100.npz, puis séparer aléatoirement les données en deux parties : un ensemble d'apprentissage (Strain) et un ensemble de validation (Svalid). L'ensemble d'apprentissage sera constitué de 2/3 des données et l'ensemble de validation de 1/3 des données. On va définir un ensemble N contenant différentes tailles d'ensemble d'apprentissage que l'on va tester. Pour chaque valeur de N, on va construire un nouvel ensemble d'apprentissage en sélectionnant les N premiers exemples de l'ensemble d'apprentissage initial. On va ensuite appliquer un algorithme d'apprentissage sur cet ensemble d'apprentissage, et utiliser la fonction de prédiction obtenue pour estimer les étiquettes des exemples de l'ensemble de validation. On va enfin calculer l'erreur quadratique moyenne sur l'ensemble de validation et sur l'ensemble d'apprentissage, et stocker ces erreurs dans les vecteurs valid_error et train_error. On va répéter ce processus pour chaque algorithme d'apprentissage que l'on a codé, en utilisant toujours les mêmes ensembles d'apprentissage et de validation. Les vecteurs valid_error et train_error deviendront alors des matrices, chaque colonne contenant les performances d'un algorithme d'apprentissage. Enfin, on va sauvegarder les résultats de l'expérience dans un fichier .npz, avec N, valid_error et train_error

4.1 Comparaison des résultats

Une fois que nous avons implémenté tout ces algorithmes , nous comparaisons leurs performances .la figure suivante donne les performances de chaque algorithme en fonction de la taille du training size. On peut aussi visualiser le temps de calcul de chacun de ces algorithmes dans la figure suivante:

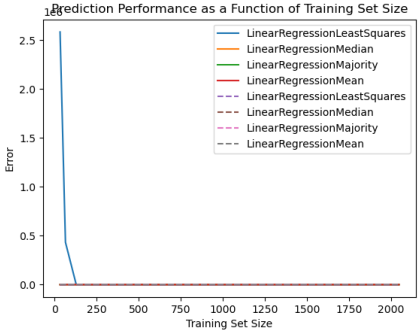

Figure 2: Performance des algos en fonction de la taille du training set

Figure 3: Le temps de calcul en fonction de la taille du training set

Nous voyons que l'erreur de la regression least Square est grande quand la taille du traing_set est petite , ce dernier peut s'expliquer par un ovearfitting.Et quand la taille atteint une certaine dimension tous les algorithmes ont à peu près le meme erreur qui est relativement faible . Le temps de cal-

cul est aussi beaucoup plus importante avec la regression Least Square et cette difference s'accentue quand la taille des données augmentent.Notons que les traits pleins reprenent l'erreur pour les données d'apprentissage.

5 Algorithmes avec régularisation

5.1 Regularisation ridge

La classe LinearRegressionRidge est une implémentation de la régression linéaire basée sur la régression de Tikhonov, également appelée régression de Ridge. Elle hérite de la classe LinearRegression et ajoute un nouveau paramètre : lambda_ridge. Ce paramètre correspond au coefficient de pénalisation non-négatif utilisé dans l'algorithme de régression de Ridge.
La méthode fit de cette classe permet d'apprendre une fonction linéaire en utilisant la régression de Ridge. Elle prend en entrée un ensemble de données d'apprentissage (X) et leurs étiquettes (y). La constante b est d'abord estimée et soustraite des étiquettes y. Le vecteur w est alors estimé en utilisant une matrice de features normalisée avec des colonnes de norme l2.
La classe LinearRegressionRidge effectue d'abord un pré-traitement des données en centrant les étiquettes y et en normalisant la matrice de features X. Elle utilise ensuite l'algorithme de régression de Ridge (ridge_regression) sur ces données pré-traitées pour estimer w_tilde. Enfin, elle effectue un post-traitement du résultat en divisant chaque élément de w_tilde par l'élément correspondant de alpha pour obtenir le vecteur w final. Les attributs w et b de l'objet sont ensuite mis à jour avec les valeurs estimées.elle est implementée dans le fichier linear_regression.py.

5.2 Les algorithmes gloutons

Les algorithmes gloutons sont une famille d'algorithmes qui cherchent à trouver une solution sous-optimale à un problème global généralement difficile en prenant des décisions locales optimales à chaque étape. Ils sont souvent utilisés pour résoudre des problèmes de recherche de solutions parcimonieuses, c'est-à-dire en sélectionnant un sous-ensemble de variables qui minimise un critère donné. Dans le cas de la régression linéaire parcimonieuse, les algorithmes gloutons sélectionnent un par un les coefficients non nuls de w en minimisant l'erreur résiduelle à chaque étape.
Deux algorithmes gloutons particulièrement courants sont l'algorithme Matching Pursuit (MP) et l'algorithme Orthogonal Matching Pursuit (OMP). L'algorithme MP est rapide mais fournit une solution moins précise que l'algorithme OMP. L'algorithme OMP est moins rapide que MP, mais fournit une solution de meilleure qualité grâce à un critère de mise à jour amélioré. Dans un contexte de ressources limitées, il est donc intéressant d'étudier ces deux algorithmes pour trouver un bon compromis entre temps de calcul et qualité de la solution.

5.2.1 MP

La classe LinearRegressionMp implémente la régression linéaire en utilisant l'algorithme Matching Pursuit (MP) pour estimer les coefficients de la fonction linéaire. Elle hérite de la classe LinearRegression et prend en compte un paramètre supplémentaire : n_iter, qui correspond au nombre d'itérations de l'algorithme MP. La méthode fit de cette classe permet d'apprendre une fonction linéaire en utilisant les données d'apprentissage X et leurs étiquettes y. Elle commence par prétraiter les données en centrant les étiquettes y et en normalisant la matrice de features X, puis utilise l'algorithme MP pour estimer le vecteur de coefficients w. Le vecteur final w est obtenu en divisant chaque élément de w_tilde par l'élément correspondant de alpha. Les attributs w et b de l'objet sont finalement mis à jour avec les valeurs estimées.

5.2.2 OMP

La classe LinearRegressionOmp est une implémentation de la régression linéaire basée sur l'algorithme Orthogonal Matching Pursuit (OMP). Elle hérite de la classe LinearRegression et ajoute un nouveau paramètre : n_iter. Ce paramètre correspond au nombre d'itérations de l'algorithme OMP.
La méthode fit de cette classe permet d'apprendre une fonction linéaire en utilisant l'algorithme OMP. Elle prend en entrée un ensemble de données d'apprentissage (X) et leurs étiquettes (y). La

constante b est d'abord estimée et soustraite des étiquettes y. Le vecteur w est alors estimé en normalisant la matrice de features X et en utilisant l'algorithme OMP sur les données pré-traitées. Le vecteur w final est obtenu en divisant chaque élément de w_tilde par l'élément correspondant de alpha. Les attributs w et b de l'objet sont ensuite mis à jour avec les valeurs estimées.

6 Intégration des algorithmes pour la régression linéaire

L'algorithme est la suivante :

Algorithm 8 Régression linéaire avec un algorithme régularisé

Require: $\mathbf{X} = [\mathbf{a}_0, ..., \mathbf{a}_{M-1}] \in \mathbb{R}^{N \times M}$, $\mathbf{y} \in \mathbb{R}^N$, hyperparamètre P (nom différent selon l'algorithme)

Estimation de b (par équation 12)

$\tilde{\mathbf{X}}, \alpha \leftarrow$ normalisation($\mathbf{X}$)

$\tilde{\mathbf{w}} \leftarrow A(\tilde{\mathbf{X}}, \tilde{\mathbf{y}}, P)$ (algorithme MP, OMP ou ridge)

$\mathbf{w} \leftarrow$ rescale($\tilde{\mathbf{w}}, \alpha$) (par équation 16)

return w

Figure 4: Algorithme régularisée.

Nous avons implementé cette algorithme pour nos 3 algorithmes (Ridge,Mp,Omp).

7 Estimation des hyperparamètres

Dans chacun des algorithmes avec régularisation présentés, au moins un paramètre doit être fixé au préalable avant d'appliquer l'algorithme. On parle d'hyperparamètre. Dans les cas ci-dessus, il s'agit dans le cas de la régression ridge, du coefficient de pénalisation ridge ; et pour MP, OMP , du nombre d'itérations n_iter . Fixer ces hyperparamètres est en général une étape difficile, qui peut demander un temps de calcul im- portant. La tâche consiste en général à essayer plusieurs valeurs possibles et à comparer les résultats pour sélectionner la valeur donnant la meilleure performance.La figure suivante donne l'erreur en fonction de la valeur de lambdas dans le cas de la regression Ridge.

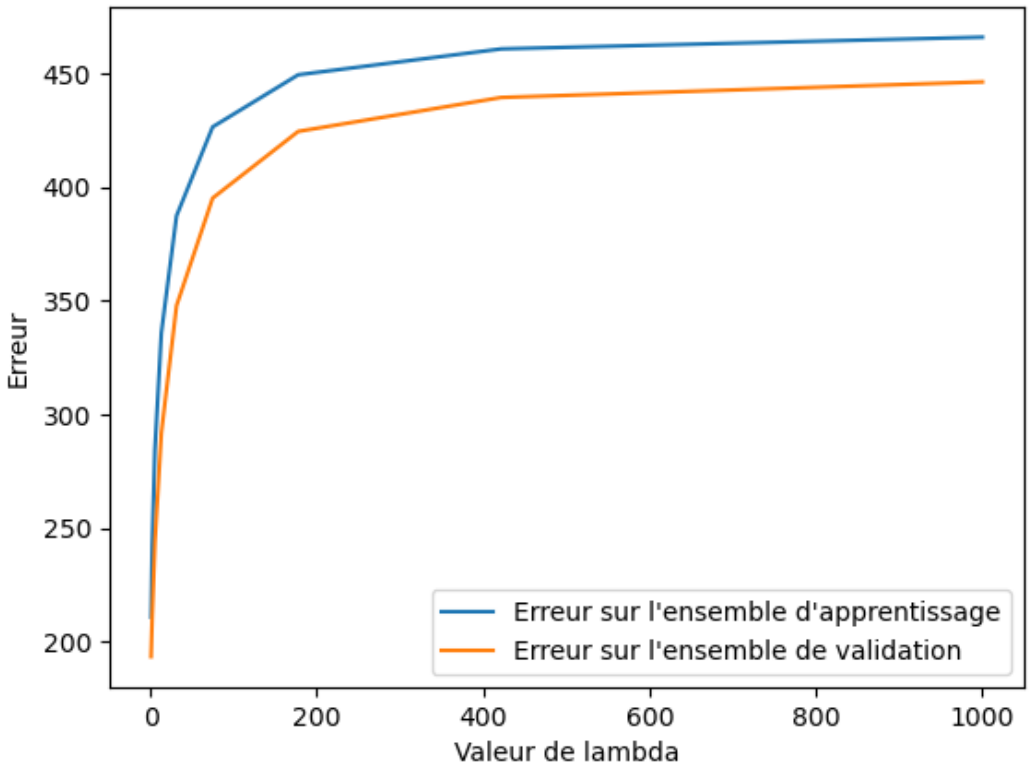

Figure 5: Erreur en fonction de lambdas.

On voit que la valeur de lambda optimale est de 420 ,avant cette valeur l'erreur d'apprentissage de que l'erreur d'entraînement augmentent. et apres cette valeur il reste constant.
Maintenant nous essayont d'entrainer notre model a partir du l'hyperparametre optimal puis nous faisons nos predictions . la figure suivante donne les performances de chaque algorithme en fonction de la taille du training size. On peut aussi visualiser le temps de calcul de chacun de ces algorithmes dans la figure suivante:

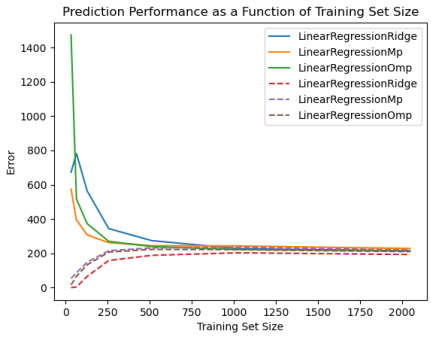

Figure 6: Performance des algos regularisés en fonction de la taille du training set

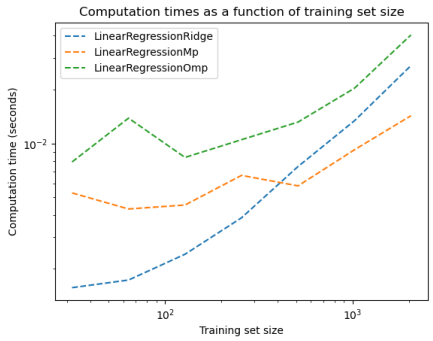

Figure 7: Le temps de calcul en fonction de la taille du training set

l'erreur de la régression ridge est plus faible que les autres peu importe la taille du training set . Quand le training set atteint une certaine dimension , tout les erreurs deviennent faible et presque égale. Le temps de calcul de la régression ridge reste relativement faible comparé aux autres algorithmes même s'il est plus élevé que celle de mp quand la taille du training set atteint une certain

dimension.
Une fois que nous avons fait tout ces étapes, on choisit notre model qui est le model qui a le plus petit erreur(régression ridge) avec la valeur du meilleur hyperparametre. l'erreur de validation est de 236.7688052597577.
Nous faisons la prediction sur les données tests avec notre model et les resultats sont enregistré dans 'Error/test_prediction_results.npy'

8 Conclusion

En conclusion, ce projet a consisté à apprendre à un ordinateur à dater une chanson en utilisant des techniques de régression linéaire et de régularisation. Nous avons exploré différentes méthodologies de programmation et avons comparé les performances de différents algorithmes élémentaires et avancés. Nous avons également mis en place une méthodologie pour estimer les hyperparamètres de ces algorithmes en utilisant la validation croisée. En fin de compte, nous avons intégré ces différentes approches pour obtenir une expérience de prédiction optimale avec des ressources limitées.