Prevision des prix des maisons a Boston

March 16, 2019

1 Introduction

Dans ce projet, j'évaluerai la performance et la puissance prédictive d'un modèle qui a été formé et testé sur des données recueillies dans des maisons de banlieue de Boston, Massachusetts. Un modèle formé à partir de ces données et considéré comme un bon ajustement pourrait ensuite être utilisé pour faire certaines prédictions sur une maison - en particulier, sur sa valeur monétaire. Ce modèle s'avérerait inestimable pour quelqu'un comme un agent immobilier qui pourrait utiliser cette information sur une base quotidienne.

L'ensemble de données pour ce projet provient du UCI Machine Learning Repository. Les données sur les logements de Boston ont été recueillies en 1978 et chacune des 506 entrées représente des données agrégées sur 14 caractéristiques des maisons de diverses banlieues de Boston, au Massachusetts. Aux fins du présent projet, les étapes de prétraitement suivantes ont été effectuées sur l'ensemble de données :

- 16 points de données ont une valeur'MEDV' de 50,0. Ces points de données contiennent probablement des valeurs manquantes ou censurées et ont été supprimés.
- 1 point de données a une valeur'RM' de 8,78. Ce point de données peut être considéré comme une valeur aberrante et a été supprimé.
- Les caractéristiques "RM", "LSTAT", "PTRATIO" et "MEDV" sont essentielles. Les autres caractéristiques non pertinentes ont été exclues.
- La fonction'MEDV' a été mise à l'échelle multiplicativement pour tenir compte de 35 années d'inflation du marché.

Le code ci-dessous sert à charger l'ensemble de données du logement Boston, ainsi que quelques-unes des bibliothèques Python nécessaires pour ce projet.

```
In [3]:  # Importer les bibliothèques nécessaires pour ce projet
         import numpy as np
         import pandas as pd
         from sklearn.model_selection import ShuffleSplit

         # Joli écran pour ordinateurs portables
         %matplotlib inline

         # Charger l'ensemble de données sur le logement de Boston
         data = pd.read_csv(r"C:\Users\acer\Desktop\housing.csv")
         prices = data['MEDV']
```

```
        features = data.drop('MEDV', axis = 1)

        # Succés
        print ('L ensemble de données sur le logement de Boston comporte {0} points de données
```

```
L ensemble de données sur le logement de Boston comporte 489 points de données avec 4 variables
```

1.1 Exploration des données

Dans cette première partie de ce projet, je ferai une étude sommaire des données sur le logement à Boston.

Puisque l'objectif principal de ce projet est de construire un modèle de travail capable de prédire la valeur des maisons, nous devrons séparer l'ensemble de données en caractéristiques et en variables cibles. Les caractéristiques,'RM','LSTAT' et'PTRATIO', nous donnent des informations quantitatives sur chaque point de données. La variable cible, "MEDV", sera la variable que nous cherchons à prédire. Ceux-ci sont enregistrés respectivement dans les caractéristiques et les prix.

1.1.1 Mise en uvre : Calculer les statistiques

Je calculerai des statistiques descriptives sur les prix des logements à Boston. Comme numpy a déjà été importé, j'utiliserai cette bibliothèque pour effectuer les calculs nécessaires. Ces statistiques seront extrêmement importantes plus tard pour analyser divers résultats de prédiction du modèle construit.

Dans la cellule de code ci-dessous, on trouvera :

Le minimum, le maximum, la moyenne, la médiane et l'écart-type de " MEDV ", qui est stocké dans les prix.

```
In [10]: # TODO : Prix minimum des données
         minimum_price = np.min(prices)

         # TODO : Prix maximum des données
         maximum_price = np.max(prices)

         # TODO : Prix moyen des données
         mean_price = np.mean(prices)

         # TODO : Prix médian des données
         median_price = np.median(prices)

         # TODO : écart-type des prix des données
         std_price = np.std(prices)

         # Il y a d'autres statistiques que vous pouvez calculer comme les quartiles
         first_quartile = np.percentile(prices, 25)
         third_quartile = np.percentile(prices, 75)
         inter_quartile = third_quartile - first_quartile
```

```
# Afficher les statistiques calculées
print ("Statistiques pour l'ensemble de données sur le logement à Boston:\n")
print ("Prix minimum: ${:,.2f}".format(minimum_price))
print ("Prix maximum: ${:,.2f}".format(maximum_price))
print ("Prix moyen : ${:,.2f}".format(mean_price))
print ("Prix médian ${:,.2f}".format(median_price))
print ("Écart-type des prix: ${:,.2f}".format(std_price))
print ("Premier quartile de prix: ${:,.2f}".format(first_quartile))
print ("Deuxième quartile de prix: ${:,.2f}".format(third_quartile))
print ("Interquartile (IQR) des prix : ${:,.2f}".format(inter_quartile))
```

```
Statistiques pour l'ensemble de données sur le logement à Boston:

Prix minimum: $105,000.00
Prix maximum: $1,024,800.00
Prix moyen : $454,342.94
Prix médian $438,900.00
Écart-type des prix: $165,171.13
Premier quartile de prix: $350,700.00
Deuxième quartile de prix: $518,700.00
Interquartile (IQR) des prix : $168,000.00
```

1.1.2 Question 1 - Observation d'un élément

Pour rappel, nous utilisons trois caractéristiques de l'ensemble de données sur le logement de Boston : RM','LSTAT' et'PTRATIO'. Pour chaque point de données (voisinage) :

- RM' est le nombre moyen de pièces dans les maisons du quartier.
- LSTAT' est le pourcentage de propriétaires dans le quartier considéré comme "classe inférieure" (travailleurs pauvres).
- PTRATIO' est le rapport élèves/enseignants dans les écoles primaires et secondaires du quartier.

1. RM

Pour une RM plus élevée, on s'attendrait à observer une MEDV plus élevée.

En effet, plus de pièces impliquerait plus d'espace, donc plus de coûts, toutes choses étant égales par ailleurs. 2. LSTAT

Pour un LSTAT plus élevé, on s'attendrait à observer un MEDV plus faible.

Les milieux sociaux d'une région dominée par des citoyens de " classe moyenne " peuvent ne pas être propices aux jeunes enfants. Elle peut aussi être relativement dangereuse par rapport à une zone dominée par les citoyens de la "classe aisée". Par conséquent, une région avec plus de citoyens de " classe moyenne " réduirait la demande et donc les prix. 3. PTRATIO

Pour un LSTAT plus élevé, on s'attendrait à observer un MEDV plus faible.

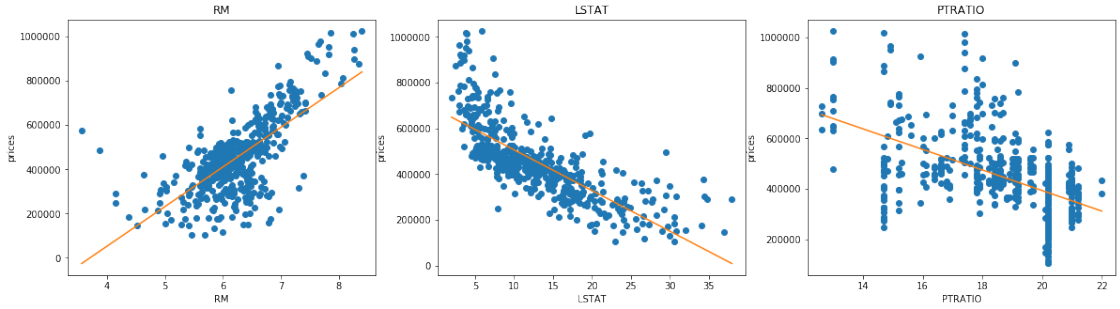
Cela s'explique par le fait qu'il y aurait un ratio enseignant/élèves plus faible, ce qui ferait en sorte que chaque élève ferait l'objet de moins d'attention, ce qui pourrait nuire à son rendement à l'école. C'est généralement le cas dans les écoles publiques/étatiques par rapport aux écoles

privées. Et les prix des maisons autour des écoles publiques sont généralement plus bas que ceux des maisons autour des écoles privées. Par conséquent, on pourrait s'attendre à un prix plus bas étant donné le taux d'encadrement élevé en raison d'une demande plus faible de maisons dans ces régions.

1.1.3 Visualisation

```
In [12]: # Utilisation de  pyplot
         import matplotlib.pyplot as plt
         plt.figure(figsize=(20, 5))

         # i: index
         for i, col in enumerate(features.columns):
             # 3 plots ici donc 1, 3
             plt.subplot(1, 3, i+1)
             x = data[col]
             y = prices
             plt.plot(x, y, 'o')
             #  Créer une ligne de régression
             plt.plot(np.unique(x), np.poly1d(np.polyfit(x, y, 1))(np.unique(x)))
             plt.title(col)
             plt.xlabel(col)
             plt.ylabel('prices')
```



1.1.4 Élaboration d'un modèle

Dans cette deuxième partie du projet, j'utiliserai les outils et les techniques nécessaires pour qu'un modèle puisse faire une prédiction. Le fait de pouvoir évaluer avec précision la performance de chaque modèle à l'aide de ces outils et techniques aide à renforcer considérablement la confiance dans les prédictions.

Mise en oeuvre : Définir une mesure de performance Il est difficile de mesurer la qualité d'un modèle donné sans quantifier sa performance par rapport à la formation et aux tests. Cela se fait généralement à l'aide d'un certain type de mesure du rendement, que ce soit en calculant un certain type d'erreur, la qualité de l'ajustement, ou une autre mesure utile. Pour ce projet,

je calculerai le coefficient de détermination R2 pour quantifier la performance du modèle. Le coefficient de détermination d'un modèle est une statistique utile dans l'analyse de régression, car il décrit souvent à quel point ce modèle est " bon " pour faire des prévisions.

Les valeurs de R2 vont de 0 à 1, ce qui représente le pourcentage de corrélation au carré entre les valeurs prévues et réelles de la variable cible. Un modèle avec un R2 de 0 ne prédit jamais la variable cible, alors qu'un modèle avec un R2 de 1 prédit parfaitement la variable cible. Toute valeur comprise entre 0 et 1 indique quel pourcentage de la variable cible, en utilisant ce modèle, peut être expliqué par les caractéristiques. Un modèle peut également se voir attribuer un R2 négatif, ce qui indique que le modèle n'est pas meilleur que celui qui prédit naïvement la moyenne de la variable cible.

```
In [16]:  # TODO: Import 'r2_score'
          from sklearn.metrics import r2_score

          def performance_metric(y_true, y_predict):
              """  Calcule et retourne le score de performance entre
                   les valeurs vraies et prévues en fonction de la métrique choisie. """

              # TODO: Calcule le score de performance entre'y_true' et'y_predict'
              score = r2_score(y_true, y_predict)

              # Retourne le score
              return score
```

1.1.5 Mise en uvre : Shuffle and Split Data

La prochaine implémentation prend l'ensemble de données de Boston et divisie les données en sous-ensembles de formation et de test. Généralement, les données sont également mélangées dans un ordre aléatoire lors de la création des sous-ensembles de formation et de test afin d'éliminer tout biais dans l'ordre de l'ensemble de données.

```
In [37]: # TODO: Import 'train_test_split'
         from sklearn.model_selection import train_test_split

         # TODO: Mélange et division des données en sous-ensembles de formation et de test.
         X_train, X_test, y_train, y_test = train_test_split(features, prices, test_size=0.2, r

         # Succés
         print ("Le partage de la formation et des tests a été un succès.")
```

```
Training and testing split was successful.
```

1.2 Évaluation du rendement du modèle

Dans cette dernière section du projet, je construirai un modèle et faire une prédiction sur l'ensemble des fonctionnalités du client en utilisant un modèle optimisé de fit_model.

1.2.1 Question 2 - Grid Search

Qu'est-ce que la Grid Search et comment l'appliquer pour optimiser un algorithme d'apprentissage ?

Réponse :

- Essentiellement, la technique de recherche par grille permet de définir une grille de paramètres qui sera recherchée à l'aide d'une validation croisée en K-fold.
- Il est important de noter que la technique de recherche par grille essaie de manière exhaustive chaque combinaison des valeurs hyper-paramétriques fournies afin de trouver le meilleur modèle.
- On peut alors trouver la plus grande précision de validation croisée qui correspond aux paramètres correspondants et qui optimise l'algorithme d'apprentissage.

1.2.2 Question 3 - Validation croisée

Qu'est-ce que la technique de formation en validation croisée en k-fold ? Quel est l'avantage de cette technique pour la recherche par grille lors de l'optimisation d'un modèle ?

Réponse :

- Résumé de la validation croisée des plis K :
  - L'ensemble de données est divisé en K "plis" de taille égale.
  - Chaque pli agit comme ensemble de test 1 fois, et agit comme ensemble d'entraînement K-1 fois.
  - La performance moyenne des tests est utilisée comme estimation de la performance hors échantillon.
  - Aussi connu sous le nom de performance contre-validée.
- Avantages de la validation croisée du pli k :
  - Estimation plus fiable de la performance hors échantillon que la répartition train/essai.
  - Réduire la variance d'un essai unique d'un fractionnement train/essai.
- Ainsi, avec les avantages de la validation croisée en k-fois, nous sommes en mesure d'utiliser la précision moyenne des tests comme référence pour décider quel est l'ensemble de paramètres le plus optimal pour l'algorithme d'apprentissage.
  - Si nous n'utilisons pas d'ensemble de validation croisée et que nous effectuons une recherche par grille, nous aurions différents ensembles de paramètres optimaux parce que, sans un ensemble de validation croisée, l'estimation du rendement hors échantillon aurait un écart élevé.
  - En résumé, sans validation croisée par le facteur k, le risque est plus élevé que la recherche par grille sélectionne des combinaisons de valeurs hyper-paramétriques qui donnent de très bons résultats sur un fractionnement d'essai en train spécifique, mais qui donnent de mauvais résultats autrement.

Limitation de la validation croisée du pli k : Il ne fonctionne pas bien lorsque les données ne sont pas uniformément réparties (p. ex. données triées).

1.2.3 Mise en uvre : Montage d'un modèle

L'implémentation finale nécessite de tout rassembler et de former un modèle à l'aide de l'algorithme de l'arbre de décision. Pour assurer la production d'un modèle optimisé, je formerai le modèle à l'aide de la technique de recherche par grille afin d'optimiser le paramètre'max_depth' de l'arbre décisionnel. Le paramètre'max_depth' peut être considéré comme le nombre de questions que l'algorithme de l'arbre de décision est autorisé à poser sur les données avant de faire une prédiction. Les arbres de décision font partie d'une classe d'algorithmes appelés algorithmes d'apprentissage supervisé.

Utilisation de GridSearchCV

```
In [33]: # TODO: Import 'make_scorer', 'DecisionTreeRegressor', and 'GridSearchCV'
         from sklearn.metrics import make_scorer
         from sklearn.tree import DecisionTreeRegressor
         from sklearn.model_selection import GridSearchCV

         def fit_model(X, y):
             """ Effectue une recherche par grille sur le paramètre'max_depth' pour un
                 régresseur de l'arbre de décision formé sur les données d'entrée[X, y]. """

             # Créer des ensembles de validation croisée à partir des données d'entraînement
             # ShuffleSplit fonctionne de façon itérative par rapport à KFOLD
             # Il permet d'économiser du temps de calcul lorsque votre ensemble de données s'a
             # X.shape[0] est le nombre total d'éléments
             cv_sets = ShuffleSplit(X.shape[0], test_size = 0.20, random_state = 0)

           # TODO : Créer un objet régresseur d'arbre de décision
             # Instantiate
             regressor = DecisionTreeRegressor(random_state=0)

             # TODO : Créer un dictionnaire pour le paramètre'max_depth' avec une plage de 1 à
             dt_range = range(1, 11)
             params = dict(max_depth=dt_range)

             # TODO : Transformer'performance_metric' en fonction de notation en utilisant'mak
             # Nous avons d'abord créé Performance_metric en utilisant R2_score
             scoring_fnc = make_scorer(performance_metric)

            # TODO : Créer l'objet de recherche par grille
             # Vous réaliseriez que nous les avons créés manuellement, y compris scoring_func
             grid = GridSearchCV(regressor, params, cv=cv_sets, scoring=scoring_fnc)

             # Ajuster l'objet de recherche par grille aux données pour calculer le modèle opt
             grid = grid.fit(X, y)

            # Retourne le modèle optimal après ajustement des données
             return grid.best_estimator_
```

Utilisation de RandomizedSearchCV

```
In [31]: # Import RandomizedSearchCV
         from sklearn.model_selection import RandomizedSearchCV

         # Créer une nouvelle fonction similaire
         def fit_model_2(X, y):
             """ Effectue une recherche par grille sur le paramètre'max_depth' pour un
                 régresseur de l'arbre de décision formé sur les données d'entrée[X, y]. """

              # Créer des ensembles de validation croisée à partir des données d'entraînement
             # ShuffleSplit fonctionne de façon itérative par rapport à KFOLD
             # Il permet d'économiser du temps de calcul lorsque votre ensemble de données s'a
             # X.shape[0] est le nombre total d'éléments
             cv_sets = ShuffleSplit(X.shape[0], test_size = 0.20, random_state = 0)

              # TODO : Créer un objet régresseur d'arbre de décision
             # Instantiate
             regressor = DecisionTreeRegressor(random_state=0)

             # TODO : Créer un dictionnaire pour le paramètre'max_depth' avec une plage de 1 à
             dt_range = range(1, 11)
             params = dict(max_depth=dt_range)

                 # TODO : Transformer'performance_metric' en fonction de notation en utilisant
             # Nous avons d'abord créé Performance_metric en utilisant R2_score
             scoring_fnc = make_scorer(performance_metric)

              # TODO : Créer l'objet de recherche par grille
             # Vous réaliseriez que nous les avons créés manuellement, y compris scoring_func
             rand = RandomizedSearchCV(regressor, params, cv=cv_sets, scoring=scoring_fnc)

             # Ajuster l'objet de recherche par grille aux données pour calculer le modèle opt
             rand = rand.fit(X, y)

            # Retourne le modèle optimal après ajustement des données
             return rand.best_estimator_
```

1.2.4 Faire des prédictions

Une fois qu'un modèle a été formé sur un ensemble donné de données, il peut maintenant être utilisé pour faire des prédictions sur de nouveaux ensembles de données d'entrée. Dans le cas d'un régresseur d'arbre décisionnel, le modèle a appris quelles sont les meilleures questions à poser au sujet des données d'entrée et peut répondre par une prévision pour la variable cible. Vous pouvez utiliser ces prédictions pour obtenir des informations sur des données dont la valeur de la variable cible n'est pas connue - par exemple des données sur lesquelles le modèle n'a pas été formé.

Optimal Model

```
In [40]: # Adapter les données de formation au modèle à l'aide d'une grille de recherche
```

```
        reg = fit_model(X_train, y_train)

        # Produisez la valeur pour'max_depth'.
        print ("Le paramètre'max_depth' est {} pour le modèle optimal.".format(reg.get_params
```

```
Le paramètre'max_depth' est 4 pour le modèle optimal.
```

Predicting Selling Prices

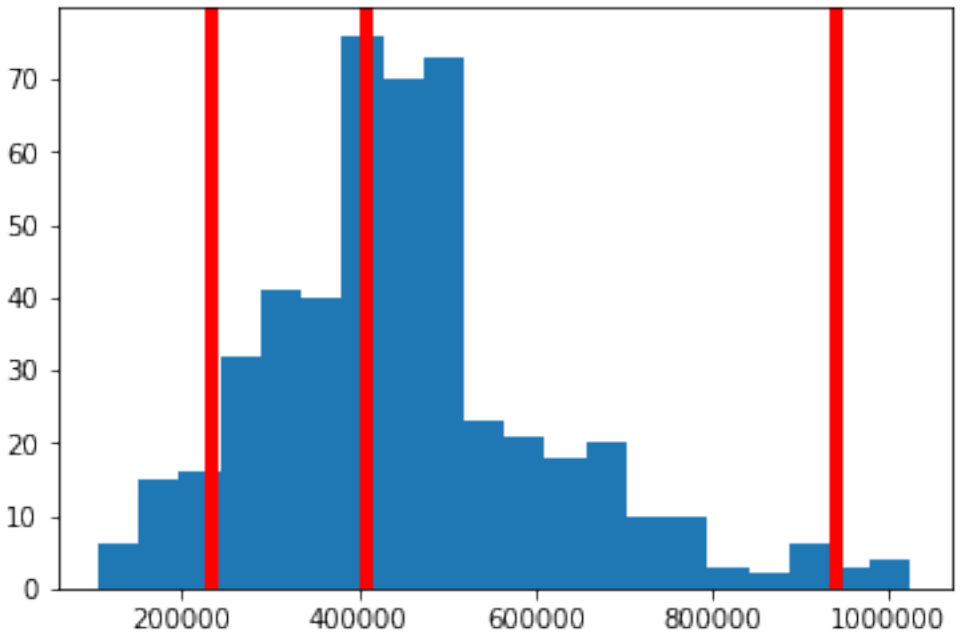
```
In [42]: # Produire une matrice pour les données des clients
         client_data = [[5, 17, 15], # Client 1
                        [4, 32, 22], # Client 2
                        [8, 3, 12]]  # Client 3

         #   Afficher les prévisions
         for i, price in enumerate(reg.predict(client_data)):
             print ("Prix de vente prévu pour la maison du client {} : ${:,.2f}".format(i+1, pr
```

```
Prix de vente prévu pour la maison du client 1 : $406,933.33
Prix de vente prévu pour la maison du client 2 : $232,200.00
Prix de vente prévu pour la maison du client 3 : $938,053.85
```

Visualisation

```
In [36]: import matplotlib.pyplot as plt
         plt.hist(prices, bins = 20)
         for price in reg.predict(client_data):
             plt.axvline(price, lw = 5, c = 'r')
```



1.3 Conclusion

- Les données recueillies dans une ville rurale peuvent ne pas s'appliquer, car les données démographiques changeraient et d'autres caractéristiques pourraient être mieux adaptées à l'ensemble de données qu'un modèle dont les caractéristiques ont été apprises à l'aide de données urbaines.
- L'algorithme d'apprentissage a appris d'un très vieil ensemble de données qui peut ne pas être pertinent parce que les données démographiques ont beaucoup changé depuis 1978.
- Il n'y a actuellement que 3 caractéristiques, il y en a d'autres qui peuvent être incluses comme le taux de criminalité, la proximité de la ville, l'accès aux transports publics et plus encore.