Rapport Machine Learning S8

RAPPORT PORTANT SUR LA CREATION D'UN MODELE DE MACHINE LEARNING APPLIQUE A UNE BASE DE DONNEE SUR LES VOITURES AFIN DE DETERMINER LE PRIX DE VENTE EN FONCTION DE DIFFERENTES VARIABLES

Bedino Gabriel

TABLE DES MATIERES

1) Traitement de la base de données

Afin de préparer les données pour l'entraînement d'un modèle de machine learning de prédiction de prix de vente de voiture, nous avons réalisé plusieurs étapes de prétraitement. L'objectif est d'assurer que les données soient dans un format exploitable, homogène et optimisé pour l'apprentissage automatique.

Chargement et exploration des données

Nous avons tout d'abord importé notre base de données via la bibliothèque `pandas` à l'aide de la commande :

```
df = pd.read_csv("carData.csv")
```

Cela nous a permis de manipuler facilement les colonnes, de visualiser les premières lignes et de commencer le traitement des variables.

Création de variables supplémentaires utiles et suppression des variables inutiles

Afin d'améliorer les capacités de prédiction du modèle, nous avons introduit une variable supplémentaire : « Car_Age », qui représente l'âge du véhicule en années. Elle est calculée à partir de l'année de fabrication (Year). L'année actuelle (2025) est utilisée comme référence. Cette transformation est plus informative pour le modèle qu'une simple année brute. Ensuite, nous avons supprimé les colonnes Car_Name (non pertinente pour l'analyse) et Year (déjà utilisée pour créer Car_Age) :

```
df["Car_Age"] = 2025 - df["Year"]
df.drop(["Car_Name", "Year"], axis=1, inplace=True)
```

Séparation des variables

Nous avons défini :

- **X** comme l'ensemble des variables explicatives (features)
- **y** comme la variable cible : `Selling_Price`, le prix de vente du véhicule

```
X = df.drop("Selling_Price", axis=1)
y = df["Selling_Price"]
```

Identification des types de colonnes et transformation des données

Il est essentiel de distinguer les colonnes numériques des colonnes catégorielles pour les transformer de manière appropriée. Nous avons donc séparé nos variables selon leur type :

```
numerical_features = ["Present_Price", "Kms_Driven", "Car_Age"]
categorical_features = ["Fuel_Type", "Seller_Type", "Transmission", "Owner"]
```

Pour standardiser les données numériques et encoder les variables catégorielles, nous avons utilisé un ColumnTransformer. Cet outil permet d'appliquer différents types de transformations à différentes colonnes.

- StandardScaler() est alors utilisé pour standardiser les données numériques (moyenne = 0, écart-type = 1), ce qui est crucial pour les modèles sensibles à l'échelle.
- OneHotEncoder(drop="first") permet de convertir les variables catégorielles en variables numériques binaires (encodage one-hot), tout en évitant la multicolinéarité en supprimant la première modalité.

```
preprocessor = ColumnTransformer(transformers=[
    ("num", StandardScaler(), numerical_features),
    ("cat", OneHotEncoder(drop="first"), categorical_features)
])
```

2) Recherche du meilleur modèle

Une fois les données préparées, nous avons procédé à une comparaison de plusieurs modèles de machine learning pour identifier celui offrant les meilleures performances en prédiction du prix de vente des voitures. Cette étape est cruciale car chaque algorithme a ses propres forces et faiblesses selon la nature des données.

Modèles comparés

Nous avons sélectionné quatre modèles de régression classiques et robustes :

- **Random Forest Regressor** : un ensemble d'arbres de décision, robuste au bruit et efficace pour modéliser des relations non linéaires.
- **Gradient Boosting Regressor** : un modèle performant basé sur une optimisation itérative des erreurs (boosting).
- **Régression linéaire** : modèle simple servant de base de comparaison.
- **Support Vector Regressor (SVR)** : puissant dans les cas complexes, mais sensible à la normalisation des données.

Chaque modèle a été intégré dans un pipeline incluant le prétraitement des données (défini précédemment) et l'algorithme proprement dit. Cela garantit que chaque modèle reçoit des données transformées de manière identique.

```
pipeline = Pipeline(steps=[
    ("preprocessing", preprocessor),
    ("regressor", reg)
])
```

Séparation des données

Avant l'entraînement, nous avons séparé notre jeu de données en un ensemble d'entraînement et un ensemble de test :

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Cela permet d'évaluer les performances du modèle sur des données jamais vues pendant l'entraînement, ce qui est indispensable pour tester la généralisation.

Évaluation des performances

Pour chaque modèle, nous avons mesuré :

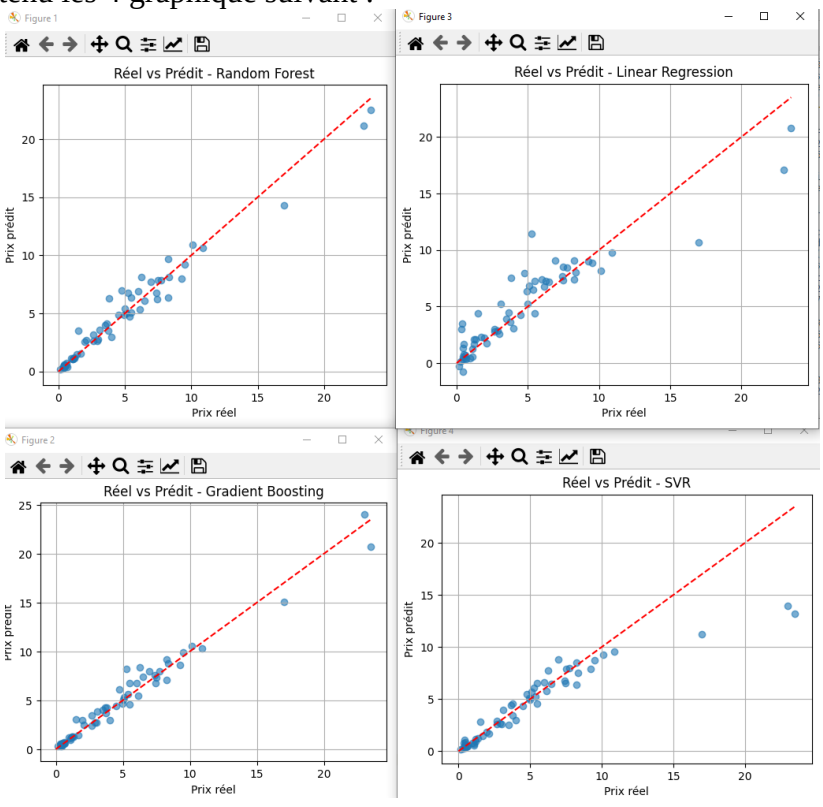
- l'erreur quadratique moyenne (MSE) : plus elle est faible, mieux c'est ;
- le coefficient de détermination (R^2) : proche de 1 = modèle très performant.
-

```
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
```

Nous avons également affiché un graphique réel vs prédit pour chaque modèle afin de visualiser leur précision.

Résultats et sélection

Nous avons obtenu les 4 graphique suivant :



Le gradient semble alors être le modèle le plus performant sur ces données (moins de dispersion autour de la droite). Cela est ensuite validé par les scores mse et r2 (respectivement 0.72 et 0.97).

3) Développement du modèle gradient et optimisation des hyperparamètres

Après avoir identifié le Gradient Boosting Regressor comme le modèle le plus performant dans notre étude comparative, nous avons poursuivi avec une phase d'optimisation des hyperparamètres. Cette étape permet de régler finement le comportement du modèle pour obtenir des performances maximales.

Pourquoi optimiser les hyperparamètres ?

Les modèles de machine learning ont des hyperparamètres, c'est-à-dire des paramètres externes au modèle, qu'il faut régler manuellement ou automatiquement (contrairement aux paramètres internes, appris pendant l'entraînement).

Dans le cas du Gradient Boosting, ces hyperparamètres influencent directement :

- la complexité du modèle (max_depth) ;
- le nombre total d'arbres (n_estimators) ;
- la vitesse d'apprentissage (learning_rate) ;
- la proportion d'échantillons utilisés à chaque étape (subsample).

Une bonne combinaison de ces valeurs peut améliorer considérablement la précision et la robustesse du modèle.

Méthodes d'optimisation utilisées

Nous avons utilisé deux approches complémentaires proposées par scikit-learn :

RandomizedSearchCV – Recherche aléatoire
Cette méthode sélectionne un sous-ensemble aléatoire de combinaisons d'hyperparamètres. Elle est plus rapide que la recherche exhaustive tout en conservant une bonne couverture de l'espace des solutions.

```
random_search = RandomizedSearchCV(
    estimator=pipeline,
    param_distributions=param_dist,
    n_iter=25,
    cv=5,
    scoring='r2',
    n_jobs=-1,
    random_state=42
)
```

Nous avons testé 25 combinaisons différentes, évaluées avec une validation croisée à 5 plis (cv=5) pour garantir la robustesse des résultats.

GridSearchCV – Recherche exhaustive
Cette méthode teste systématiquement toutes les combinaisons possibles d'un petit ensemble défini d'hyperparamètres. Elle est plus longue mais peut être plus précise dans un espace réduit.

```
grid_search = GridSearchCV(
    estimator=pipeline,
    param_grid=param_grid,
    cv=5,
    scoring='r2',
    n_jobs=-1
)
```

Résultats et sélection du modèle optimisé

À l'issue des recherches, nous avons comparé les résultats des deux méthodes. Le meilleur modèle a été sélectionné à partir de random_search.best_estimator_ ou grid_search.best_estimator_, selon les performances observées.

```
best_model = random_search.best_estimator_
```

Nous testons ensuite le MSE et R2 de la même manière que précédemment (on trace aussi un graphique) et on remarque que le modèle s'est légèrement amélioré.

```
✅ Meilleurs paramètres (Grid Search) : {'regressor__learning_rate': 0.01, 'regressor__
max_depth': 3, 'regressor__n_estimators': 100, 'regressor__subsample': 0.6}
📊 Gradient Boosting Optimisé
MSE: 0.7003452815140742
R² Score: 0.9695972428701692
```

4) Création de l'interface utilisateur

Afin de rendre le modèle de prédiction accessible à des utilisateurs non techniques, nous avons développé une interface utilisateur intuitive et interactive à l'aide de la bibliothèque Streamlit. Cette application web permet de saisir des caractéristiques de véhicule et d'obtenir instantanément une estimation du prix de vente.

Objectifs de l'interface

L'objectif de l'interface est double :

- Faciliter l'utilisation du modèle de prédiction pour tout utilisateur (professionnel ou particulier) ;
- Améliorer l'expérience utilisateur en proposant une interface claire, responsive, et enrichie de fonctionnalités utiles.

Fonctions principales intégrées

Saisie manuelle des caractéristiques du véhicule
L'utilisateur peut renseigner :

- le prix neuf du véhicule (Present_Price) ;
- le kilométrage (Kms_Driven) ;
- l'âge de la voiture (Car_Age) ;
- le type de carburant (Essence, Diesel, CNG) ;
- le type de vendeur (Particulier ou Concessionnaire) ;
- le type de transmission (Manuelle ou Automatique) ;
- le nombre de propriétaires précédents.

Ces champs sont tous sélectionnables via des menus déroulants ou des zones de saisie numériques simples.

Prédiction automatique du prix de vente
À partir des données saisies, l'interface utilise le modèle optimisé Gradient Boosting Optimisé créer précédemment (chargé via un fichier.pkl) pour calculer le prix estimé de la voiture, affiché dynamiquement à l'écran.

Calcul de la perte de valeur
Pour donner davantage de sens à la prédiction, l'interface calcule aussi la différence entre le prix neuf et le prix estimé, permettant à l'utilisateur de visualiser la dépréciation estimée du véhicule.

Historique des prédictions
Chaque prédiction effectuée est enregistrée automatiquement dans une table affichée à l'écran. Cette fonctionnalité permet à l'utilisateur de comparer plusieurs véhicules ou plusieurs scénarios de vente.

Téléchargement des résultats
L'utilisateur peut télécharger l'historique complet des prédictions au format CSV pour archivage, impression ou partage.

Résultat

CENTRALE LYON
ENISE

Prédiction du prix de vente d'une voiture

Informations sur la voiture

Prix neuf de la voiture (en k €)

5,40 − +

Kilométrage (Kms Driven)

20000 − +

Âge de la voiture (en années)

12 − +

Type de carburant

Petrol

Type de vendeur

Dealer

Transmission

Manual

Nombre de propriétaires précédents

0

Prédire le prix

Prix estimé : 3.17k €

Perte estimée de valeur : 2.23k €

Historique des prédictions

	Kms_Driven	Car_Age	Fuel_Type	Seller_Type	Transmission	Owner	Prix prédit	Perte de valeur
0	20000	12	Petrol	Dealer	Manual	0	3.1659	2.2341

Télécharger l'historique au format CSV