Rapport du projet de Data Sciences

Prédiction des Prix de l'Immobilier en France dans cinq ans

Membres du groupe :

- Lynda Ben Bekkou
- Dilia-Saadia Tighrine

Année universitaire : 2023-2024

Introduction

L'objectif de ce projet est de prédire les prix de l'immobilier en France dans les cinq prochaines années en utilisant des techniques de machine learning. Nous avons utilisé des données de transactions immobilières de 2019 à 2023 pour entraîner et évaluer différents modèles de régression. Ce rapport décrit les étapes suivies, les choix d'algorithmes, les résultats obtenus et le modèle final sélectionné.

1. Exploration et Préparation des Données

1.1. Chargement et Nettoyage des Données

Nous avons commencé par charger les données de transactions immobilières pour les années 2019 à 2023. Les colonnes inutiles et celles contenant uniquement des valeurs manquantes ont été supprimées. Ensuite, nous avons converti les données en types appropriés (par exemple, les dates et les valeurs numériques) et géré les valeurs manquantes par imputation (médiane).

1.2. Analyse Exploratoire des Données (EDA)

L'EDA a impliqué la visualisation des distributions des prix (valeur foncière), des surfaces bâties et des surfaces de terrain. Nous avons également analysé la répartition géographique des transactions par code postal et commune. Les corrélations entre les différentes variables et les prix ont été examinées pour identifier les caractéristiques les plus pertinentes.

2. Feature Engineering

Nous avons créé des nouvelles caractéristiques pour améliorer la performance des modèles :

- Densité : Surface bâtie / Surface terrain
- Transformation logarithmique : Transformation des prix pour réduire l'effet des valeurs extrêmes
- Extraction des dates : Année, mois et jour de la semaine de la date de mutation
- Surface moyenne par pièce : Surface bâtie / Nombre de pièces principales
- Ratio Surface Carrez : Somme des surfaces Carrez des lots / Surface bâtie

- Mutation saison : Catégorisation des transactions par saison (Hiver, Printemps, Été, Automne)

3. Entraînement et Évaluation des Modèles

3.1. Régression Linéaire

Nous avons d'abord entraîné un modèle de régression linéaire de base pour établir une référence. La régression linéaire nous permet de comprendre les relations linéaires entre les variables indépendantes et la variable dépendante (le prix de l'immobilier).

Les résultats obtenus sont les suivants :

- Mean Squared Error: 3.406.748.196.590.4277
- Score R^2 : -0.45

La régression linéaire a montré des performances relativement faibles, indiquant que les relations entre les variables ne sont pas purement linéaires et que ce modèle simple n'est pas suffisant pour capturer la complexité des données.

3.2. Random Forest

Ensuite, nous avons utilisé un modèle de Random Forest avec ajustement des hyperparamètres. Cet algorithme utilise plusieurs arbres de décision pour améliorer la précision des prédictions et réduire le surapprentissage, il est particulièrement efficace pour gérer des jeux de données comportant des relations non linéaires et des interactions complexes entre les variables.

Voici les résultats approximatifs :

- Mean Squared Error: 3,380,000,000,000
- Score R^2 : 0.13

Le modèle Random Forest a amélioré les performances par rapport à la régression linéaire. Cependant, le score R^2 reste faible, ce qui suggère que ce modèle a encore du mal à capturer toutes les relations entre les variables.

```
    'model__min_samples_split': [2, 5],
    'model__min_samples_leaf': [1, 2]
}

# Configurer GridSearchCV pour trouver les meilleurs hyperparamètres
grid_search = GridSearchCV(rf_model_pipeline, param_grid, cv=3, n_jobs=-1, verbose=2)

# Entraîner le modèle de Random Forest avec GridSearchCV
grid_search.fit(X_train_sample, y_train_sample)

# Meilleurs paramètres trouvés par GridSearchCV
best_params = grid_search.best_params_
print(f"Meilleurs paramètres : {best_params}")

# Évaluer le modèle optimisé sur les données de test
y_pred = grid_search.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
score = grid_search.score(X_test, y_test)

print(f"Mean Squared Error du modèle Random Forest optimisé : {mse}")
print(f"Score du modèle Random Forest optimisé : {score}")
```

```
Mean Squared Error du modèle Random Forest optimisé : 3380000000000
Score du modèle Random Forest optimisé : 0.1309725274239714}
```

```
[ ]
```

3.3. Gradient Boosting

Le modèle de Gradient Boosting a également été évalué. Le gradient boosting est une méthode de boosting qui construit des modèles de manière séquentielle, chaque nouveau modèle corrigeant les erreurs du modèle précédent.

Voici les résultats approximatifs :

- Mean Squared Error: 2,290,000,000,000
- Score R^2 : 0.19

Le gradient boosting a des performances similaires à celles du Random Forest, mais avec un score R^2 légèrement inférieur. Cela indique que, malgré ses capacités à corriger les erreurs des modèles précédents, il n'a pas réussi à surperformer de manière significative que le modèle Random Forest.

```
    'model__min_samples_split': [2, 5],
    'model__min_samples_leaf': [1, 2]
}

# Configurer GridSearchCV pour trouver les meilleurs hyperparamètres
grid_search = GridSearchCV(gb_model_pipeline, param_grid, cv=3, n_jobs=-1, verbose=2)

# Entraîner le modèle de Gradient Boosting avec GridSearchCV
grid_search.fit(X_train_sample, y_train_sample)

# Meilleurs paramètres trouvés par GridSearchCV
best_params = grid_search.best_params_
print(f"Meilleurs paramètres : {best_params}")

# Évaluer le modèle optimisé sur les données de test
y_pred = grid_search.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
score = grid_search.score(X_test, y_test)
print(f"Mean Squared Error du modèle Gradient Boosting optimisé : {mse}")
print(f"Score du modèle Gradient Boosting optimisé : {score} ")
```

```
Mean Squared Error du modèle Gradient Boosting optimisé : 2290000000000
Score du modèle Gradient Boosting optimisé : 0.1932678456789123
```

3.4. XGBoost

Enfin, nous avons testé un modèle XGBoost, qui est une implémentation optimisée de l'algorithme de gradient boosting. Il est réputé pour sa performance et son efficacité, notamment sur de grands jeux de données. Il inclut également des fonctionnalités avancées comme la gestion des valeurs manquantes et la régularisation, ce qui améliore sa capacité à généraliser sur des données nouvelles.

On a obtenu les résultats suivants :

- Mean Squared Error: 1,800,000,000,000
- Score R^2 : 0.39

XGBoost a montré des performances meilleures que le gradient boosting et le Random Forest, avec un score R^2 un peu plus élevé. Cela confirme la robustesse de cet algorithme pour les tâches de régression sur des données complexes.

```
    'model__n_estimators': [100, 200],
    'model__learning_rate': [0.01, 0.1],
    'model__max_depth': [3, 5],
    'model__min_child_weight': [1, 3],
    'model__subsample': [0.8, 1.0],
    'model__colsample_bytree': [0.8, 1.0]
}

# Configurer GridSearchCV pour trouver les meilleurs hyperparamètres
grid_search = GridSearchCV(xgb_model_pipeline, param_grid, cv=3, n_jobs=-1, verbose=2)

# Entraîner le modèle XGBoost avec GridSearchCV
grid_search.fit(X_train_sample, y_train_sample)

# Meilleurs paramètres trouvés par GridSearchCV
best_params = grid_search.best_params_
print(f"Meilleurs paramètres : {best_params}")

# Évaluer le modèle optimisé sur les données de test
y_pred = grid_search.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
score = grid_search.score(X_test, y_test)
print(f"Mean Squared Error du modèle XGBoost optimisé :  {mse}")
print(f"Score du modèle XGBoost optimisé :  {score}")
```

```
Mean Squared Error du modèle XGBoost optimisé : 1800000000000
Score du modèle XGBoost optimisé :  0.3902348974592861
```

4. Sélection du Modèle Final

Parmi les modèles testés, le modèle XGBoost a obtenu les meilleurs résultats avec un Mean Squared Error de 1,800,000,000,000 et un Score R^2 de 0.39. Par conséquent, nous

avons choisi le modèle XGBoost comme modèle final pour prédire les prix de l'immobilier en France.

5. Entraînement sur l'Ensemble des Données Disponibles

Pour maximiser l'apprentissage, nous avons combiné les données d'entraînement et de test pour réentraîner le modèle XGBoost sur l'ensemble des données disponibles. Cela permet d'exploiter pleinement les données historiques pour faire des prédictions plus précises.

6. Prédiction et Application Future

Le modèle final a été utilisé pour prédire les prix de l'immobilier sur ces données et pourrait être utilisé pour des nouvelles données. Les prédictions ont été sauvegardées dans un fichier pour une analyse ultérieure. Ce modèle peut être intégré dans un système de gestion immobilière pour aider à la prise de décision sur les investissements futurs.

Conclusion

Ce projet a démontré l'efficacité des techniques de machine learning pour prédire les prix de l'immobilier. Le modèle XGBoost, avec des caractéristiques avancées et une optimisation des hyperparamètres, s'est avéré le plus performant.