8INF867 – FONDAMENTAUX DE L'APPRENTISSAGE AUTOMATIQUE - HIVER 2022
Professeur Julien Maitre, Ph.D.

UNIVERSITÉ DU QUÉBEC À CHICOUTIMI
Département d'Informatique et de Mathématique

TRAVAIL PRATIQUE 2 - Linear Regression

BALB10020109 - Becaye Baldé
MEDM28048006 - Marcelo Medeiros de Vasconcellos
LIMN28030005 - Nicolas Lim
MIRV14100108 - Victor Mira

Sommaire

Introduction

Ce rapport fait partie des travaux pratiques 2 du sujet 8INF867 - FONDAMENTAUX DE L'APPRENTISSAGE AUTOMATIQUE - AUTOMNE 2022 de l'Université du Québec à Chicoutimi. Nous verrons dans ce rapport la description du jeu de données, les fonctions de traitement des données, un pipeline de formation et de test des modèles et la comparaison des résultats sur les modèles LinearRegression, Ridge, Lasso et ElasticNet. Le programme a été développé en python et peut être exécuté soit en ligne de commande, soit via le notebook jupyter. Pour exécuter le programme en ligne de commande, nous suggérons la création d'un environnement virtuel pour installer les bibliothèques, comme indiqué ci-dessous.

Comme indiqué dans les consignes du TP2, dans ce travail, nous avons utilisé au maximum les codes du TP1, les rendant plus réutilisables et préparant le code à être pour nos travaux futurs.

Lancement du programme

Pour créer l'environnement de développement:

Dans le dossier, lancer la commande: `python -m venv venv`

Pour activer l'environnement virtuel:

Windows: `.\venv\Scripts\activate`

Mac/Linux: `source venv\bin\activate`

Pour installer la bibliothèque requise: `pip install -r requirements.txt`

Désactiver l'environnement virtuel: `deactivate`

Pour lancer le LinearRegression (programme principal), il suffit de lancer le fichier : `python main.py`

Vous pouvez également exécuter le programme à partir de Jupyter Notebook via le fichier : `main.ipynb`

1. Ensemble de données

Le jeu de données choisi est celui des prix des maisons de la ville de Perth en Australie. Ces données peuvent être consultées à l'adresse suivante : https://www.kaggle.com/datasets/syuzai/perth-house-prices .

L'ensemble de données est fourni dans un fichier CSV, comprend 33656 cas et 19 attributs, dont 'price' est la cible, qui indique le prix de la maison.

Caractéristiques	Type	Données manquantes	Description
address	String	-	Adresse de la rue
suburb	String	-	Nom de la banlieue
price	Number	-	Prix à la date de la dernière vente
bedrooms	Number	-	Nombre de chambres
bathrooms	Number	-	Nombre de salles de bain

garage	Number	2478	Nombre de garages/espaces pour voitures
land_area	Number	-	Superficie du terrain en mètres carrés
floor_area	Number	-	Surface de plancher en mètres carrés
build_year	Number	3155	Année de construction de la maison
cbd_dist	Number	-	Distance par rapport au quartier central des affaires
nearest_stn	String	-	Gare ferroviaire la plus proche
nearest_stn_dist	Number	-	Distance de la gare la plus proche
date_sold	String	-	Date de la dernière vente de la propriété
postcode	Number	-	Code postal de la banlieue
latitude	Number	-	Latitude de l'adresse
longitude	Number	-	Longitude de l'adresse
nearest_sch	String	-	L'école la plus proche où l'on peut appliquer l'ATAR
nearest_sch_dist	Number	-	Distance par rapport à l'école la plus proche où l'on peut appliquer le système ATAR
nearest_sch_rank	Number	10952	Rang de l'école appliquant l'ATAR

2. Traitement des données

2.1. Supprimer les données non importantes

address: Cette fonctionnalité a été supprimée car les données d'adresse sont trop granulaires.
nearest_sch_rank: Supprimé car environ 33% des données sont manquantes dans cette caractéristique.

2.2. Les données manquantes

2.2.1. Traitement des caractéristique garage et build_year

La caractéristique "garage" a 2478 données manquantes et la caractéristique "année de construction" a 3155 données manquantes, dans ce cas nous avons remplacé les données manquantes par la médiane.

Captures d'écran des extraits de code

```
def process_missing_values(self):
    log('Dataset : Processing missing values')
    for num_field in self.num_fields:
        imputer = SimpleImputer(missing_values=np.nan, strategy='median')
        imputer.fit(self.df[[num_field]])
        self.df[num_field] = imputer.transform(self.df[[num_field]])
    for field in self.categorical_fields:
        self.df[field] = self.df[field].replace(np.nan, 'Not Informed')
```

2.3. Les données catégorielles

2.3.1. Traitement des caractéristiques: nearest_sch, nearest_stn, suburb

Dans le cas de données catégorielles qui ont plus de deux classes, nous calculons le degré de représentativité de chaque classe, pour les cas où la classe représente moins de 5% du nombre total de cas, nous changeons le nom de la classe en "Others" .

Captures d'écran des extraits de code

```
def reduce_dimensionality_categorical_data(self, percent=0.05):
    log('Dataset : Reducing the dimensionality of categorical data')
    for field in self.categorical_fields:
        data_temp = self.df[field].value_counts().to_dict()
        total = self.df.shape[0]
        for value in data_temp:
            perc = data_temp[value] / total
            if perc <= percent:
                self.df[field] = self.df[field].replace(value, 'Others')
```

2.4. La date de vente

La date de vente est définie comme Mois-Année, pour le traitement de celle-ci nous inversons la position informant Année-Mois, supprimons le trait d'union et la convertissons en un nombre entier.

Captures d'écran des extraits de code

```
def traitement_date_sold(self):
    log('Dataset : Date Sold Traitement')
    date_sold = []
    for date in self.df['date_sold']:
        date = date.strip().split('-')
        date_sold.append(int(date[1] + date[0]))
    self.df['date_sold'] = date_sold
```

3. Pipeline d'entraîner et tester un modèle de régression

Après le traitement des données, nous avons réalisé un pipeline avec le préprocesseur contenant les algorithmes PolynomialFeatures (pour la génération de caractéristiques polynomiales et d'interaction), StandardScaler (réduction de la dimensionnalité) pour le traitement des données numériques et OneHotEncoder pour le traitement des données catégorielles. Après l'exécution du préprocesseur, le pipeline exécute l'algorithme PCA puis l'algorithme de régression.

L'algorithme de régression varie entre les quatre étudiés en classe, à savoir : LinearRegression, Ridge, Lasso et ElasticNet.

Captures d'écran des extraits de code

```python
class Algorithms:
    def __init__(self, X_train, X_test, y_train, y_test, num_fields, categorical_fields): ...

    def load_preprocessor(self):
        numeric_transformer = Pipeline(steps=[
            ('PolynomialFeatures', PolynomialFeatures(degree=2)),
            ('StandardScaler', StandardScaler())
            ])
        categorical_transformer = Pipeline(steps=[
            ('OneHotEncoder', OneHotEncoder(handle_unknown='ignore'))])

        self.preprocessor = ColumnTransformer(
            transformers=[
                ('numeric_transformer', numeric_transformer, self.num_fields),
                ('categorical_transformer', categorical_transformer, self.categorical_fields)])

    def load_pipeline(self):
        self.abbreviation = f'Pipe{self.abbreviation}'
        self.name = f'Pipeline(preprocessor + PCA + {self.name})'
        self.model = Pipeline(
            steps=[('preprocessor', self.preprocessor),
                   ('PCA', PCA()),
                   ('regressor', self.model)])
        log(F'Algorithms : Loaded {self.name}')
```

Captures d'écran du programme s'exécutant avec le notebook jupyter

La capture d'écran ci-dessous montre l'exécution de l'algorithme de régression linéaire sans le pipeline.

```
%%time
algo.load_linear_regression()
algo.fit_predict()
evaluation.add(algo)
```

```
2022-10-06 13:41:50,459 [INFO] Algorithms : Loaded LinearRegression
2022-10-06 13:41:50,484 [INFO] Algorithms : Fit and Predict LinearRegression
2022-10-06 13:41:50,486 [INFO] Evaluation : LinearRegression : R2 Score = 0.5845611986540789
2022-10-06 13:41:50,488 [INFO] Evaluation : LinearRegression : Mean Squared Error = 54152959725.30613
2022-10-06 13:41:50,494 [INFO] SelectKBestPipeline : Starting
2022-10-06 13:41:50,495 [INFO] SelectKBestPipeline : Executing Min Max Scaler
2022-10-06 13:41:50,509 [INFO] SelectKBestPipeline : Executing SelectKBest
2022-10-06 13:41:52,352 [INFO] Evaluation : Selected Features : [floor_area, cbd_dist, nearest_stn_dist, postcode]
2022-10-06 13:41:52,354 [INFO] Evaluation : Add : LinearRegression
```

```
CPU times: user 2.57 s, sys: 628 ms, total: 3.2 s
Wall time: 1.9 s
```

La capture d'écran ci-dessous montre l'exécution de l'algorithme de régression linéaire avec le pipeline.

```
%%time
algo.load_linear_regression()
algo.load_pipeline()
algo.fit_predict()
evaluation.add(algo)
```

```
2022-10-06 13:41:56,725 [INFO] Algorithms : Loaded LinearRegression
2022-10-06 13:41:56,727 [INFO] Algorithms : Loaded Pipeline(preprocessor + PCA + LinearRegression)
2022-10-06 13:41:57,140 [INFO] Algorithms : Fit and Predict Pipeline(preprocessor + PCA + LinearRegression)
2022-10-06 13:41:57,143 [INFO] Evaluation : Pipeline(preprocessor + PCA + LinearRegression) : R2 Score = 0.7495577779
944411
2022-10-06 13:41:57,147 [INFO] Evaluation : Pipeline(preprocessor + PCA + LinearRegression) : Mean Squared Error = 32
645452273.22292
2022-10-06 13:41:57,158 [INFO] SelectKBestPipeline : Starting
2022-10-06 13:41:57,159 [INFO] SelectKBestPipeline : Executing Min Max Scaler
2022-10-06 13:41:57,169 [INFO] SelectKBestPipeline : Executing SelectKBest
2022-10-06 13:41:59,125 [INFO] Evaluation : Selected Features : [floor_area, cbd_dist, nearest_stn_dist, postcode]
2022-10-06 13:41:59,127 [INFO] Evaluation : Add : Pipeline(preprocessor + PCA + LinearRegression)
```

```
CPU times: user 3.68 s, sys: 792 ms, total: 4.47 s
Wall time: 2.4 s
```

4. Ensemble des outils

4.1 Sélection des caractéristiques les plus représentatives

Dans notre base de données, nous possédons de nombreuses caractéristiques différentes. Il faut donc choisir les caractéristiques les plus représentatives pour permettre d'avoir un meilleur apprentissage des données.

Captures d'écran des extraits de code

```
def select_k_best(self, algo):
    n_components = 4
    X = pd.concat([algo.X_train, algo.X_test])
    y = pd.concat([algo.y_train, algo.y_test])
    X = X[algo.num_fields]
    k_best = SelectKBestPipeline(X, y, n_components)
    k_best.execute_min_max_scaler()
    k_best.execute_selectkbest()
    log(f"Evaluation : Selected Features : [{', '.join(k_best.selected_features)}]")
    return k_best.selected_features
```

4.2 Calcul du coefficient de détermination R2

Pour vérifier si notre choix de caractéristiques est optimal, nous allons calculer le coefficient de détermination. Ce coefficient permet de mesurer le pourcentage d'explications des caractéristiques sur le prix de vente d'une maison.

Captures d'écran des extraits de code

```
def get_r2_score(self, algo):
    r2_scr = r2_score(algo.y_test, algo.y_pred)
    log(f"Evaluation : {algo.name} : R2 Score = {r2_scr}")
    return r2_scr
```

4.3 Calcul de l'erreur

Pour vérifier que nous n'avons pas de valeurs absurdes, nous calculons aussi l'erreur entre les données prédit et les données obtenus avec notre régression linéaire. Pour ce faire, nous utilisons la méthode de calcul d'erreur quadratique moyenne.

Captures d'écran des extraits de code

```
def get_mean_squared_error(self, algo):
    mean_squared_err = mean_squared_error(algo.y_test, algo.y_pred)
    log(f"Evaluation : {algo.name} : Mean Squared Error = {mean_squared_err}")
    return mean_squared_err
```

4.4 Affichage des graphiques

Pour mieux étudier nos résultats, nous avons fait un outil pour afficher les résultats sous forme de graphique.

Captures d'écran des extraits de code

```
def save_comparison_chart(self, field):
    log(f'Evaluation : Saving chart {field}')
    import matplotlib.pyplot as plt
    new_df = pd.DataFrame.from_dict(self.algorithms)
    sns_plot = sns.barplot(
        data=new_df.reset_index(), x='abbreviation', y=field)
    sns_plot.figure.savefig(f"chart/comparison_{field}.png")
    plt.close()
```

5. Comparaison des algorithmes de régression linéaire

Pour la comparaison entre les algorithmes, nous exécutons les algorithmes de régression linéaire (Régression linéaire, Ridge et Lasso) séparément, puis nous les exécutons dans le pipeline.

- Linear Regression (LR)
- Ridge (R)
- Lasso (L)
- ElasticNet (EN)
- Pipeline(preprocessor + PCA + LinearRegression) (PipeLR)
- Pipeline(preprocessor + PCA + Ridge) (PipeR)
- Pipeline(preprocessor + PCA + Lasso) (PipeL)
- Pipeline(preprocessor + PCA + ElasticNet) (PipeEN)

5.1. Comparaison

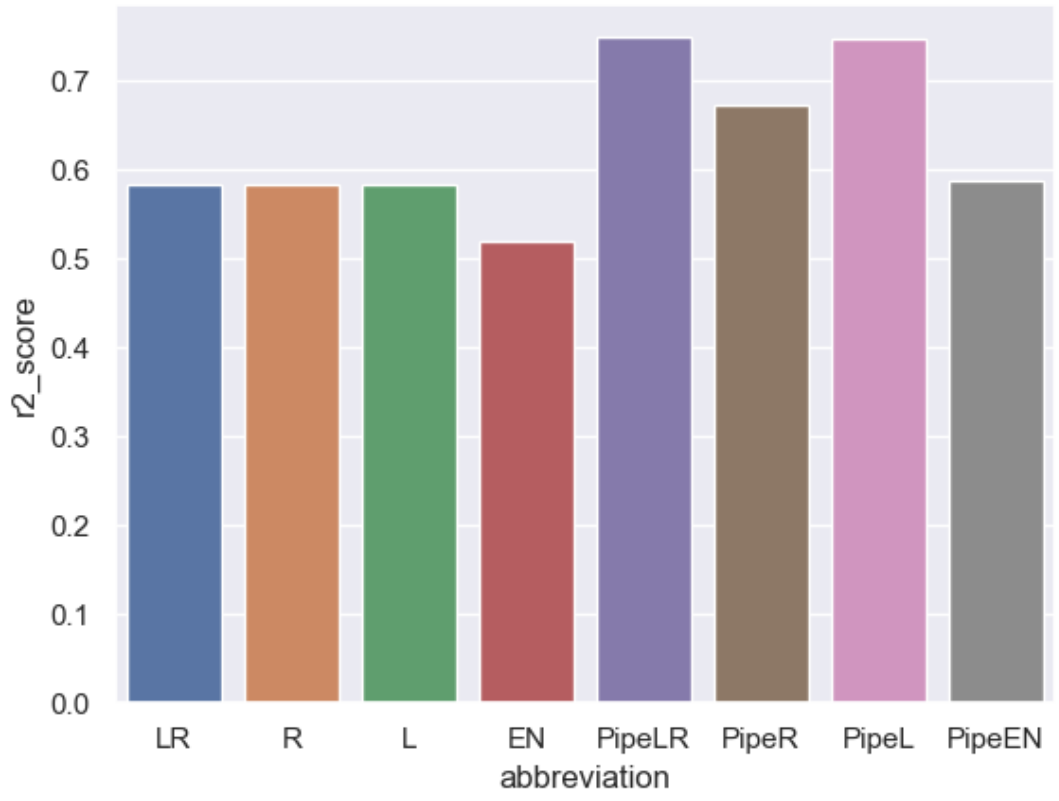
Nous avons utilisé plusieurs algorithmes afin de trouver le plus performant. Les algorithmes de régression linéaire simplement appliqués ont un faible R2 score d'environ 0.58.

En revanche, il s'est avéré qu'en utilisant un pipeline combinant le preprocessing, le PCA et l'algorithme de régression linéaire (Régression linéaire, Ridge, Lasso et ElasticNet), la précision est nettement améliorée.

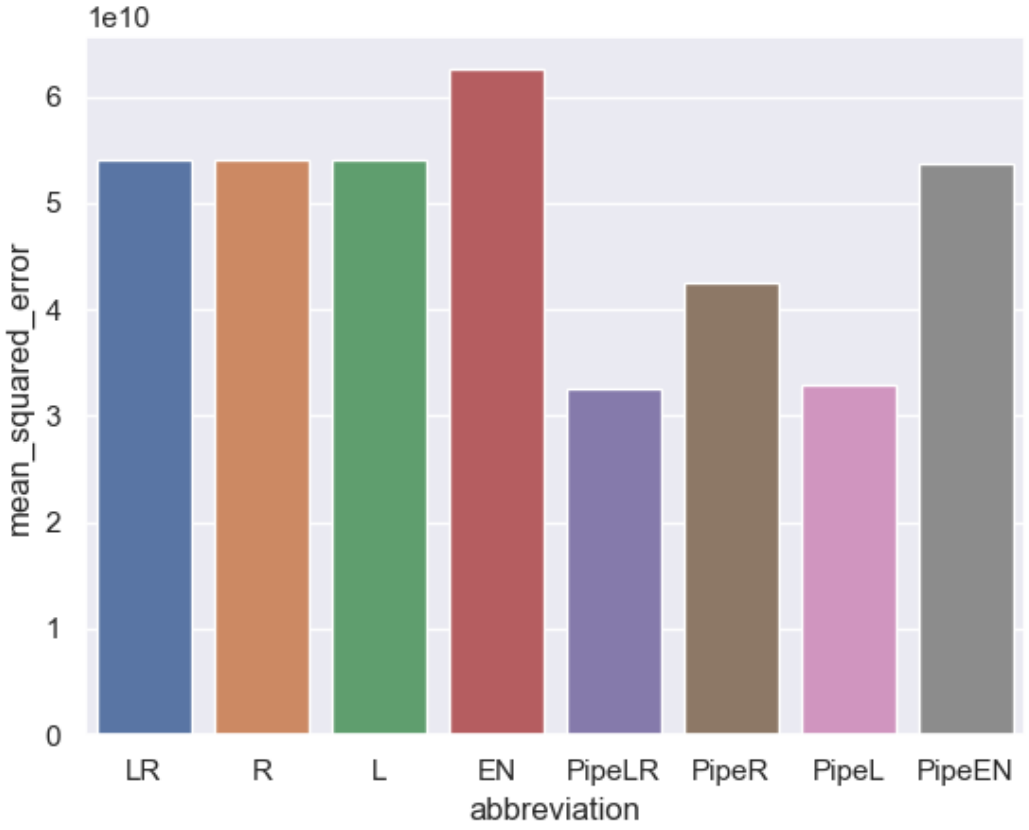
	abbreviation	name	r2_score	mean_squared_error
0	LR	LinearRegression	0.584561	5.415296e+10
1	R	Ridge	0.584576	5.415098e+10
2	L	Lasso	0.584562	5.415290e+10
3	EN	ElasticNet	0.519885	6.258364e+10
4	PipeLR	Pipeline(preprocessor + PCA + LinearRegression)	0.749558	3.264545e+10
5	PipeR	Pipeline(preprocessor + PCA + Ridge)	0.674370	4.244629e+10
6	PipeL	Pipeline(preprocessor + PCA + Lasso)	0.747471	3.291748e+10
7	PipeEN	Pipeline(preprocessor + PCA + ElasticNet)	0.588298	5.366580e+10

Le graphe suivant nous permet de mieux visualiser les performances de chaque algorithme notamment le R2 Score et le Mean Squared Error.

Graphique de comparaison du R2 score



Graphique de comparaison du mean squared error



L'objectif étant de maximiser le R2 score et de réduire le mean squared error on peut en conclure que pour cet ensemble de données, les algorithmes de régression les plus performant sont la régression linéaire et Lasso (avec utilisation de la pipeline) et le moins performant est Elastic Net (avec ou sans pipeline).