Atelier2_SGHURI_EZZEBDI

March 26, 2023

0.1 Importation des donnes et des libraries

[1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import OrdinalEncoder
import warnings
warnings.filterwarnings("ignore")
```

[2]:
```
data = pd.read_csv('C:/Users/21261/Desktop/Machine learning/autos_mpg.csv')
```

[3]:
```
data.head()
```

[3]:
```
    MPG  CYLINDERS  DISPLACEMENT   HP  WEIGHT  ACCELERATION  YEAR ORIGIN  \
0  18.0          8         307.0  130    3504          12.0    70    USA
1  15.0          8         350.0  165    3693          11.5    70    USA
2  18.0          8         318.0  150    3436          11.0    70    USA
3  16.0          8         304.0  150    3433          12.0    70    USA
4  17.0          8         302.0  140    3449          10.5    70    USA

                        NAME
0  chevrolet chevelle malibu
1          buick skylark 320
2         plymouth satellite
3              amc rebel sst
4                ford torino
```

0.2 Data description and preprocessing

[4]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 398 entries, 0 to 397
Data columns (total 9 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   MPG            398 non-null    float64
 1   CYLINDERS      398 non-null    int64
```

```
 2   DISPLACEMENT  398 non-null    float64
 3   HP            398 non-null    object
 4   WEIGHT        398 non-null    int64
 5   ACCELERATION  398 non-null    float64
 6   YEAR          398 non-null    int64
 7   ORIGIN        398 non-null    object
 8   NAME          398 non-null    object
dtypes: float64(3), int64(3), object(3)
memory usage: 28.1+ KB
```

[5]:
```python
# la colonne Hp est de type string
data['HP'] = pd.to_numeric(data['HP'],errors = 'coerce')
```

[6]:
```python
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 398 entries, 0 to 397
Data columns (total 9 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   MPG           398 non-null    float64
 1   CYLINDERS     398 non-null    int64
 2   DISPLACEMENT  398 non-null    float64
 3   HP            392 non-null    float64
 4   WEIGHT        398 non-null    int64
 5   ACCELERATION  398 non-null    float64
 6   YEAR          398 non-null    int64
 7   ORIGIN        398 non-null    object
 8   NAME          398 non-null    object
dtypes: float64(4), int64(3), object(2)
memory usage: 28.1+ KB
```

[7]:
```python
data.isna().sum()
```

[7]:
```
MPG             0
CYLINDERS       0
DISPLACEMENT    0
HP              6
WEIGHT          0
ACCELERATION    0
YEAR            0
ORIGIN          0
NAME            0
dtype: int64
```

[8]:
```python
# la colonne HP contient 6 valeurs manquantes
```

[9]:
```python
data.dropna(axis = 0, inplace = True)
```

[10]:
```
data.isna().sum()
```

[10]:
```
MPG             0
CYLINDERS       0
DISPLACEMENT    0
HP              0
WEIGHT          0
ACCELERATION    0
YEAR            0
ORIGIN          0
NAME            0
dtype: int64
```

[11]:
```
data['NAME'].value_counts()
```

[11]:
```
amc matador            5
ford pinto             5
toyota corolla         5
toyota corona          4
amc hornet             4
                      ..
buick skyhawk          1
chevrolet monza 2+2    1
ford mustang ii        1
pontiac astro          1
chevy s-10             1
Name: NAME, Length: 301, dtype: int64
```

[12]:
```
data.drop('NAME', axis = 1, inplace=True)
```

[13]:
```
data.head(2)
```

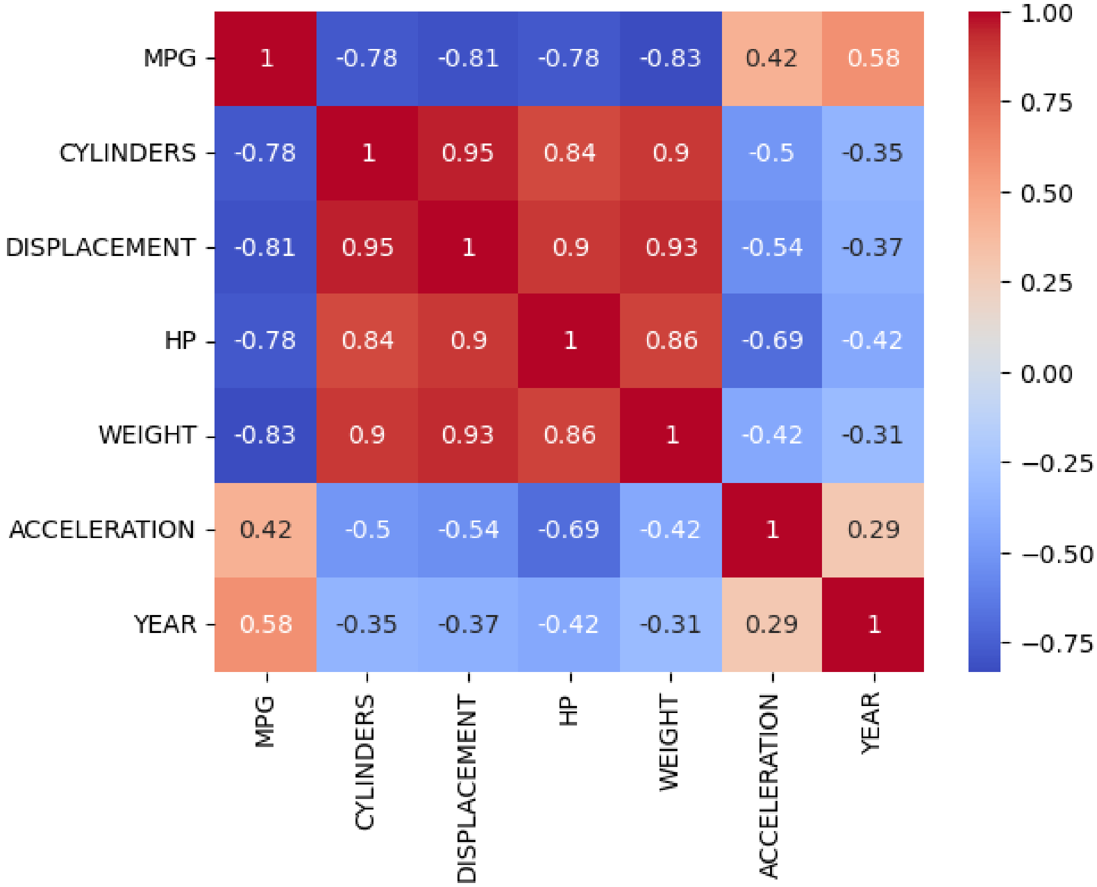
[13]:

	MPG	CYLINDERS	DISPLACEMENT	HP	WEIGHT	ACCELERATION	YEAR	ORIGIN
0	18.0	8	307.0	130.0	3504	12.0	70	USA
1	15.0	8	350.0	165.0	3693	11.5	70	USA

[14]:
```
# matrice de correlation
```

[15]:
```
sns.heatmap(data.corr(),annot = True,cmap = 'coolwarm')
```

[15]: <AxesSubplot:>



[16]:
```python
# encoder la colonne Origin
```

[17]:
```python
data['ORIGIN'].value_counts()
```

[17]:
```
USA        245
Japan       79
Germany     68
Name: ORIGIN, dtype: int64
```

[18]:
```python
#Encodage par la methode de Dummies
data_dummies = pd.get_dummies(data,prefix=['ORIGIN'], columns =
['ORIGIN'], drop_first=True)
data_dummies.head()
```

[18]:
```
    MPG  CYLINDERS  DISPLACEMENT     HP  WEIGHT  ACCELERATION  YEAR  \
0  18.0          8         307.0  130.0    3504          12.0    70
1  15.0          8         350.0  165.0    3693          11.5    70
2  18.0          8         318.0  150.0    3436          11.0    70
3  16.0          8         304.0  150.0    3433          12.0    70
```

```
4  17.0          8         302.0  140.0    3449           10.5    70

   ORIGIN_Japan  ORIGIN_USA
0             0           1
1             0           1
2             0           1
3             0           1
4             0           1
```

[19]:
```python
# Ordinal encoder
data_toenc = data[['ORIGIN']]
data_toenc = data_toenc.to_numpy()
enc = OrdinalEncoder()
enc_fitted = enc.fit(data_toenc)
encoded = enc_fitted.transform(data_toenc)
data_ordinal = data
data_ordinal['ORIGIN'] = encoded
data_ordinal.head()
```

[19]:

	MPG	CYLINDERS	DISPLACEMENT	HP	WEIGHT	ACCELERATION	YEAR	ORIGIN
0	18.0	8	307.0	130.0	3504	12.0	70	2.0
1	15.0	8	350.0	165.0	3693	11.5	70	2.0
2	18.0	8	318.0	150.0	3436	11.0	70	2.0
3	16.0	8	304.0	150.0	3433	12.0	70	2.0
4	17.0	8	302.0	140.0	3449	10.5	70	2.0

0.3 Regression lineaire

0.3.1 Holdout

[20]:
```python
# D'abord on va importer les packages nécessaires.
#import necessary packages
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from sklearn import linear_model
from sklearn.metrics import r2_score
```

[21]:
```python
X = data_dummies.drop(['MPG'], axis = 1).values[:,:]
Y = data_dummies['MPG']
X1 = data_ordinal.drop(['MPG'], axis = 1).values[:,:]
Y1 = data_ordinal['MPG']
```

[22]:
```python
X_train, X_test, y_train, y_test = train_test_split(X,Y, test_size=0.25)
```

[23]:
```python
model = linear_model.LinearRegression()
model.fit(X_train, y_train)
yhat_LR = model.predict(X_test)
```

```
# évaluer les predictions
mse_hold = mean_squared_error(y_test, yhat_LR)
print('MSE : %.3f' %mse_hold)
mae_hold = mean_absolute_error(y_test,yhat_LR)
print('MAE : %.3f' %mae_hold)
r2_hold = r2_score(y_test,yhat_LR)
print('R^2 : %.3f' %r2_hold)
```

```
MSE : 8.790
MAE : 2.188
R^2 : 0.841
```

[24]:
```
X_train, X_test, y_train, y_test = train_test_split(X1,Y1, test_size=0.25)
```

[25]:
```
model = linear_model.LinearRegression()
model.fit(X_train, y_train)
yhat_LR = model.predict(X_test)
# évaluer les predictions
score4 = mean_squared_error(y_test, yhat_LR)
print('MSE : %.3f' %score4)
score5 = mean_absolute_error(y_test,yhat_LR)
print('MAE : %.3f' %score5)
score6 = r2_score(y_test,yhat_LR)
print('R^2 : %.3f' %score6)
```

```
MSE : 8.411
MAE : 2.284
R^2 : 0.835
```

[26]:
```
comparaison = pd.DataFrame({'Regression': ['MSE','MAE','R^2'], 'Dummies':␣
↪[mse_hold, mae_hold,r2_hold], 'Ordinal encoder ': [score4, score5,score6]},
                           index = [0,1,2])
```

[27]:
```
comparaison
```

[27]:

	Regression	Dummies	Ordinal encoder
0	MSE	8.789838	8.410859
1	MAE	2.187542	2.284051
2	R^2	0.841382	0.835369

0.3.2 Feature selection

[28]:
```
X = data_dummies.drop(['MPG'], axis = 1).values[:,:]
Y = data_dummies['MPG']
```

[29]:
```
X_train, X_test, y_train, y_test = train_test_split(X,Y, test_size=0.25)
```

0.3.3 Backward

[30]:
```
from mlxtend.feature_selection import SequentialFeatureSelector as sfs
model_LR = linear_model.LinearRegression()
sfs = sfs(model_LR, k_features='best',scoring='r2', cv=5, forward = False)
sfs = sfs.fit(X_train, y_train) #trouver les meilleurs attributs
#transformer la matrice original et ne laisser que les attributs sélectionnés
X_train_sfs = sfs.transform(X_train)
# transforation de la matrice de test
X_test_sfs = sfs.transform(X_test)
# Entrainer le modèle de régression sur les données d'entrainements réduites
model_LR.fit(X_train_sfs, y_train)
#Prédiction sur les données de test
y_pred = model_LR.predict(X_test_sfs)
```

[31]:
```
#évaluer les predictions
r2_back =r2_score(y_test, y_pred)
print('R2 : %.3f' %r2_back)
mse_back = mean_squared_error(y_test, y_pred)
print('MSE : %.3f' %mse_back)
mae_back = mean_absolute_error(y_test,y_pred)
print('MAE : %.3f' %mae_back)
```

```
R2 : 0.828
MSE : 8.880
MAE : 2.293
```

[32]:
```
sfs_results = pd.DataFrame.from_dict(sfs.get_metric_dict()).T
sfs_results.sort_values(by='avg_score', ascending=False, inplace=True)
sfs_results
```

[32]:
```
                 feature_idx  \
6        (0, 1, 2, 3, 5, 7)
3                  (3, 5, 7)
7     (0, 1, 2, 3, 5, 6, 7)
5           (1, 2, 3, 5, 7)
4              (2, 3, 5, 7)
8  (0, 1, 2, 3, 4, 5, 6, 7)
2                     (3, 5)
1                       (3,)

                                            cv_scores avg_score  \
6  [0.8423524002524013, 0.8172866605234208, 0.815…   0.79777
3  [0.8342208093131547, 0.8121875823597373, 0.818…  0.797265
7  [0.8421624291901656, 0.8158339923075457, 0.814…  0.797255
5  [0.8401236570823194, 0.8210920513363291, 0.809…  0.796552
4  [0.8344688628047742, 0.8134367308145256, 0.816…  0.794856
8  [0.8312474291949358, 0.81650855772972, 0.81567…  0.794578
```

```
2  [0.8325107254371447, 0.7875839530077737, 0.823…  0.788179
1  [0.6587187606884604, 0.6644010602011845, 0.777…  0.672732

             feature_names  ci_bound   std_dev   std_err
6       (0, 1, 2, 3, 5, 7)  0.046546  0.036214  0.018107
3                (3, 5, 7)  0.039859  0.031011  0.015506
7    (0, 1, 2, 3, 5, 6, 7)   0.04655  0.036217  0.018109
5          (1, 2, 3, 5, 7)  0.045558  0.035445  0.017723
4             (2, 3, 5, 7)  0.042911  0.033386  0.016693
8 (0, 1, 2, 3, 4, 5, 6, 7)  0.044385  0.034533  0.017267
2                   (3, 5)  0.050521  0.039307  0.019654
1                     (3,)  0.069897  0.054383  0.027191
```

0.3.4 Forward

[33]:
```
X = data_dummies.drop(['MPG'], axis = 1).values[:,:]
Y = data_dummies['MPG']
```

[34]:
```
X_train, X_test, y_train, y_test = train_test_split(X,Y, test_size=0.25)
```

[35]:
```
from mlxtend.feature_selection import SequentialFeatureSelector as sfs
model_LR = linear_model.LinearRegression()
sfs = sfs(model_LR, k_features='best',scoring='r2', cv=5, forward = True)
sfs = sfs.fit(X_train, y_train) #trouver les meilleurs attributs
#transformer la matrice original et ne laisser que les attributs sélectionnés
X_train_sfs = sfs.transform(X_train)
# transforation de la matrice de test
X_test_sfs = sfs.transform(X_test)
# Entrainer le modèle de régression sur les données d'entrainements réduites
model_LR.fit(X_train_sfs, y_train)
#Prédiction sur les données de test
y_pred = model_LR.predict(X_test_sfs)
```

[36]:
```
#évaluer les predictions
r2_for =r2_score(y_test, y_pred)
print('R2 : %.3f' %r2_for)
mse_for = mean_squared_error(y_test, y_pred)
print('MSE : %.3f' %mse_for)
mae_for = mean_absolute_error(y_test,y_pred)
print('MAE : %.3f' %mae_for)
```

```
R2 : 0.812
MSE : 10.595
MAE : 2.517
```

0.3.5 Comparaison

[37]:
```
comparaison = pd.DataFrame({'Regression': ['MSE','MAE','R^2'], 'Holdout':␣
↪[mse_hold, mae_hold,r2_hold],
                            'Backward ': [mse_back, mae_back,r2_back],
                           'Forward':[mse_for , mae_for, r2_for]
                           },
                       index = [0,1,2])
```

[38]:
```
comparaison
```

[38]:

	Regression	Holdout	Backward	Forward
0	MSE	8.789838	8.879765	10.595218
1	MAE	2.187542	2.293390	2.516968
2	R^2	0.841382	0.828178	0.812158

0.4 Tree regressor

0.4.1 Holdout

[39]:
```
X_train, X_test, y_train, y_test = train_test_split(X,Y, test_size=0.25)
```

[40]:
```
from sklearn.tree import DecisionTreeRegressor
```

[41]:
```
from sklearn.tree import DecisionTreeRegressor

model = DecisionTreeRegressor(random_state=0)
model.fit(X_train,y_train)
#prédire test set
yhat_tree = model.predict(X_test)
#évaluer les predictions
mse = mean_squared_error(y_test, yhat_tree)
mae = mean_absolute_error(y_test, yhat_tree)
r2 = r2_score(y_test ,yhat_tree)

print('r2 : %.3f' %r2)
print('MSE : %.3f' %mse)
print('MAE : %.3f' %mae)
```

```
r2 : 0.765
MSE : 13.633
MAE : 2.605
```

0.4.2 cross validation

[42]:
```
from sklearn.model_selection import cross_val_score
```

[43]:
```
from sklearn.model_selection import cross_val_score
dt = DecisionTreeRegressor(random_state=0)
```

```python
dt_fit = dt.fit(X, Y)
dt_scores = cross_val_score(dt_fit, X,Y, cv = 5,
scoring='r2')
r2_cv = abs(np.mean(dt_scores))
print("r2 score: {}".format(abs(np.mean(dt_scores))))

dt_scores = cross_val_score(dt_fit, X, Y, cv = 5,
scoring='neg_mean_absolute_error')
mae_cv = abs(np.mean(dt_scores))
print("MAE score: {}".format(abs(np.mean(dt_scores))))

dt_scores = cross_val_score(dt_fit, X, Y, cv = 5,
scoring='neg_mean_squared_error')
mse_cv = abs(np.mean(dt_scores))
print("MSE score: {}".format(abs(np.mean(dt_scores))))
```

```
r2 score: 0.5619098728423182
MAE score: 2.8103472898409607
MSE score: 16.599129503407987
```

0.4.3 cross validation et grid search

[44]:
```python
#cv and grid search
from sklearn.model_selection import GridSearchCV
model = DecisionTreeRegressor()
parameter_space = {
 'min_samples_split': range(2, 10),
 'max_depth': (20, 30, 50)
}
reg = GridSearchCV(model, parameter_space, scoring =
'r2', cv=5, n_jobs=-1)

reg.fit(X, Y)
# Afficher les resultats

y_pred = reg.best_estimator_.predict(X_test)

mse_cv_gr = mean_squared_error(y_test, y_pred)
mae_cv_gr = mean_absolute_error(y_test, y_pred)
r2_cv_gr = r2_score(y_test ,y_pred)

print('r2 : %.3f' %r2_cv_gr)
print('MSE : %.3f' %mse_cv_gr)
print('MAE : %.3f' %mae_cv_gr)
```

```
r2 : 0.984
MSE : 0.937
MAE : 0.610
```

[45]:
```
comparaison = pd.DataFrame({'Tree Regressor': ['MSE','MAE','R^2'], 'Holdout':␣
↪[mse, mae, r2], 'Cv=5': [mse_cv, mae_cv,r2_cv],
                            'Cv + grid search':[mse_cv_gr,mae_cv_gr,r2_cv_gr]},
                        index = [0,1,2])
```

[46]:
```
comparaison
```

[46]:

	Tree Regressor	Holdout	Cv=5	Cv + grid search
0	MSE	13.632551	16.599130	0.937430
1	MAE	2.605102	2.810347	0.610187
2	R^2	0.765003	0.561910	0.983841

0.5 SVR

0.5.1 Holdout

[58]:
```
from sklearn.svm import SVR
```

[59]:
```
X_train, X_test, y_train, y_test = train_test_split(X,Y, test_size=0.25)
```

[60]:
```
from sklearn.model_selection import cross_val_score
model = SVR(C=1.0, epsilon=0.2)
model.fit(X_train, y_train)
yhat_SVR = model.predict(X_test)
#évaluer les predictions
score = mean_squared_error(y_test, yhat_SVR)
print('MSE',score)
score1 = mean_absolute_error(y_test, yhat_SVR)
print('MAE',score1)
score2 = r2_score(y_test, yhat_SVR)
print('R^2',score2)
```

```
MSE 16.73275067740287
MAE 2.8201017257036693
R^2 0.7016423283360842
```

0.5.2 cross validation

[65]:
```
from sklearn.model_selection import cross_val_score
svr = SVR(C=1.0, epsilon=0.2)
svr_fit = svr.fit(X, Y)
svr_scores = cross_val_score(svr_fit, X,Y, cv = 5,
scoring='r2')
r2_cv = abs(np.mean(svr_scores))
print("r2 score: {}".format(abs(np.mean(svr_scores))))

svr_scores = cross_val_score(svr_fit, X, Y, cv = 5,
scoring='neg_mean_absolute_error')
```

```
mae_cv = abs(np.mean(svr_scores))
print("MAE score: {}".format(abs(np.mean(svr_scores))))

svr_scores = cross_val_score(svr_fit, X, Y, cv = 5,
scoring='neg_mean_squared_error')
mse_cv = abs(np.mean(svr_scores))
print("MSE score: {}".format(abs(np.mean(svr_scores))))
```

```
r2 score: 0.3305546785820734
MAE score: 3.665585958748609
MSE score: 24.81875438278366
```

0.6 CV + Grid search

[62]:
```
#cv and grid search
from sklearn.model_selection import GridSearchCV
model = SVR()
parameter_space = {
 'gamma': (1e-2, 1e-4),
 'C': (1, 10),
 'epsilon':[0.1,0.5,0.3]
}
reg = GridSearchCV(model, parameter_space, scoring = 'r2', cv=5, n_jobs=-1)

reg.fit(X, Y)
# Afficher les resultats

y_pred = reg.best_estimator_.predict(X_test)

mse_cv_gr = mean_squared_error(y_test, y_pred)
mae_cv_gr = mean_absolute_error(y_test, y_pred)
r2_cv_gr = r2_score(y_test ,y_pred)

print('r2 : %.3f' %r2_cv_gr)
print('MSE : %.3f' %mse_cv_gr)
print('MAE : %.3f' %mae_cv_gr)
```

```
r2 : 0.817
MSE : 10.257
MAE : 1.952
```

[63]:
```
comparaison = pd.DataFrame({'SVR': ['MSE','MAE','R^2'], 'Holdout': [score,␣
 ↪score1, score2], 'Cv=5': [mse_cv, mae_cv,r2_cv],
                           'Cv + grid search':[mse_cv_gr,mae_cv_gr,r2_cv_gr]},
                      index = [0,1,2])
```

[64]:
```
comparaison
```

[64]:
```
   SVR    Holdout       Cv=5  Cv + grid search
0  MSE  16.732751  24.998166         10.256693
1  MAE   2.820102   3.683036          1.951732
2  R^2   0.701642   0.325592          0.817115
```

0.7 Feature Selection

[53]:
```python
from mlxtend.feature_selection import ExhaustiveFeatureSelector as EFS
svr = SVR()
efs = EFS(estimator=svr, # The Ml model
min_features=1,
max_features=6,
scoring='r2', cv=5)
efs = efs.fit(X, Y)
print('Best accuracy score: %.2f' % efs.best_score_) # best_score_ shows
print('Best subset (corresponding names):', efs.best_feature_names_)
```

```
Features: 246/246

Best accuracy score: 0.42
Best subset (corresponding names): ('2', '5')
```

[54]:
```python
# Show the performance of each subset of features
efs_results = pd.DataFrame.from_dict(efs.get_metric_dict()).T
efs_results.sort_values(by='avg_score', ascending=False, inplace=True)
efs_results
```

[54]:
```
      feature_idx                                          cv_scores  \
23         (2, 5)  [0.6791716719581615, 0.5544253723394681, 0.672…
59      (1, 2, 5)  [0.6695575454668838, 0.649822655048941, 0.7613…
15         (1, 2)  [0.644364146796536, 0.6347377296806291, 0.7586…
131  (1, 2, 4, 5)  [0.6608763207632918, 0.655603215323884, 0.7656…
94   (0, 1, 2, 5)  [0.6592465073329628, 0.6540207303525469, 0.765…
..            …                                                 …
88      (4, 5, 6)  [-0.3897425704429669, -0.24634408117230122, 0…
161  (4, 5, 6, 7)  [-0.44522663537797924, -0.27368989870670135, 0…
34         (5, 7)  [-0.483136837661466, -0.35571739428175264, -0…
33         (5, 6)  [-0.49243269971229653, -0.359630648022528, -0…
91      (5, 6, 7)  [-0.5465217348094007, -0.36417898599912224, -0…

    avg_score feature_names  ci_bound   std_dev   std_err
23   0.424481        (2, 5)  0.543178  0.422611  0.211305
59   0.410888     (1, 2, 5)  0.673806  0.524244  0.262122
15   0.400919        (1, 2)  0.673826   0.52426   0.26213
131  0.397869  (1, 2, 4, 5)  0.697322   0.54254   0.27127
94   0.395534  (0, 1, 2, 5)  0.699468   0.54421  0.272105
..        …             …         …         …         …
88  -0.811805     (4, 5, 6)   1.67538  1.303503  0.651751
```

```
161 -0.841714  (4, 5, 6, 7)  1.690719  1.315437  0.657718
34  -0.866348        (5, 7)  1.658259  1.290182  0.645091
33  -0.874541        (5, 6)  1.665713  1.295981   0.64799
91  -0.903259     (5, 6, 7)  1.686206  1.311925  0.655963

[246 rows x 7 columns]
```

[55]:
```
#Transformer le dataset pour garder seulement les attributs sélectionnés
X_train_efs = efs.transform(X_train)
X_test_efs = efs.transform(X_test)
```

[56]:
```
# Entrainer le modèle de régression sur les données d'entrainements réduites
model_SVR = SVR()
model_SVR.fit(X_train_efs, y_train)
#Prédiction sur les données de test
y_pred = model_SVR.predict(X_test_efs)
#évaluer les predictions
score =r2_score(y_test, y_pred)
print('R2 : %.3f' %score)
score1 = mean_squared_error(y_test, y_pred)
print('MSE',score1)
score2 = mean_absolute_error(y_test, y_pred)
print('MAE',score2)
```

```
R2 : 0.742
MSE 14.956254566435975
MAE 2.7153659652706796
```

[57]:
```
from sklearn.pipeline import Pipeline
clf = Pipeline([
 ('feature_selection', EFS(SVR(), scoring='r2',
max_features=6, cv=5)),
 ('classification', SVR())
])
clf.fit(X_train, y_train)
clf.score(X_test, y_test)
```

```
Features: 246/246
```

[57]:
```
0.7421848428384162
```

1 MLP Regressor

[79]:
```
X_train, X_test, y_train, y_test = train_test_split(X,Y, test_size=0.25)
```

[80]:
```
from sklearn.neural_network import MLPRegressor
```

1.0.1 Holdout

[81]:
```
model = MLPRegressor(hidden_layer_sizes={100, 200,10}, activation='relu',
solver='adam', max_iter= 5000)
model.fit(X_train, y_train)
yhat_MLP = model.predict(X_test)
#évaluer les predictions
score = mean_squared_error(y_test, yhat_MLP)
print('MSE : %.3f' %score)

score1 = mean_absolute_error(y_test, yhat_MLP)
print('MAE : %.3f' %score1)

score2 = r2_score(y_test, yhat_MLP)
print('R2 : %.3f' %score2)
```

```
MSE : 20.934
MAE : 3.306
R2 : 0.672
```

1.0.2 Cross validation

[82]:
```
from sklearn.model_selection import cross_val_score
model = MLPRegressor(hidden_layer_sizes={100, 200,10}, activation='relu',
solver='adam', max_iter= 5000)
mlp_fit = model.fit(X, Y)
mlp_scores = cross_val_score(mlp_fit, X,Y, cv = 5,
scoring='r2')
r2_cv = abs(np.mean(mlp_scores))
print("r2 score: {}".format(abs(np.mean(mlp_scores))))

mlp_scores = cross_val_score(mlp_fit, X, Y, cv = 5,
scoring='neg_mean_absolute_error')
mae_cv = abs(np.mean(mlp_scores))
print("MAE score: {}".format(abs(np.mean(mlp_scores))))

mlp_scores = cross_val_score(mlp_fit, X, Y, cv = 5,
scoring='neg_mean_squared_error')
mse_cv = abs(np.mean(mlp_scores))
print("MSE score: {}".format(abs(np.mean(mlp_scores))))
```

```
r2 score: 0.5485438712736739
MAE score: 2.6040047070959824
MSE score: 41.46024327833085
```

1.0.3 Cross validation + grid search

[83]:
```
#cv and grid search
from sklearn.model_selection import GridSearchCV
model =  MLPRegressor()
parameter_space = {
 'hidden_layer_sizes': [{100},{100, 200,50}, {100,100}],
 'solver': ['adam', 'sgd'],
 'batch_size':[200,50],
 'learning_rate':['0.001, 0.0001','adaptive']
}
reg = GridSearchCV(model, parameter_space, scoring = 'r2', cv=5, n_jobs=-1)

reg.fit(X, Y)
# Afficher les resultats

y_pred = reg.best_estimator_.predict(X_test)

mse_cv_gr = mean_squared_error(y_test, y_pred)
mae_cv_gr = mean_absolute_error(y_test, y_pred)
r2_cv_gr = r2_score(y_test ,y_pred)

print('r2 : %.3f' %r2_cv_gr)
print('MSE : %.3f' %mse_cv_gr)
print('MAE : %.3f' %mae_cv_gr)
```

```
r2 : 0.812
MSE : 12.011
MAE : 2.485
```

[84]:
```
comparaison = pd.DataFrame({'MLP Regressor': ['MSE','MAE','R^2'], 'Holdout':␣
↪[score, score1, score2], 'Cv=5': [mse_cv, mae_cv,r2_cv],
                            'Cv + grid search':[mse_cv_gr,mae_cv_gr,r2_cv_gr]},
                       index = [0,1,2])
```

[85]:
```
comparaison
```

[85]:

	MLP Regressor	Holdout	Cv=5	Cv + grid search
0	MSE	20.933571	41.460243	12.010743
1	MAE	3.306018	2.604005	2.485042
2	R^2	0.672295	0.548544	0.811978

1.0.4 Feature selection

[86]:
```
from sklearn.neural_network import MLPRegressor
from mlxtend.feature_selection import ExhaustiveFeatureSelector as EFS
mlp = MLPRegressor(hidden_layer_sizes={100, 200,10}, activation='relu',
solver='adam', max_iter= 5000)
```

```python
efs = EFS(estimator=mlp, # The Ml model
min_features=1,
max_features=3,
scoring='r2', cv=5)
efs = efs.fit(X, Y)
print('Best accuracy score: %.2f' % efs.best_score_) # best_score_ shows
print('Best subset (corresponding names):', efs.best_feature_names_)
```

```
Features: 92/92

Best accuracy score: 0.69
Best subset (corresponding names): ('1', '2', '5')
```

[87]:
```python
# Show the performance of each subset of features
efs_results = pd.DataFrame.from_dict(efs.get_metric_dict()).T
efs_results.sort_values(by='avg_score', ascending=False, inplace=True)
efs_results
```

[87]:
```
   feature_idx                                          cv_scores avg_score  \
59   (1, 2, 5)  [0.7788563158467994, 0.7469187789860989, 0.768…  0.691992
44   (0, 2, 5)  [0.7224569025596292, 0.6941792375855631, 0.780…  0.664177
66   (1, 4, 5)  [0.7870603309704198, 0.7840161028069, 0.772393…  0.656205
76   (2, 4, 5)  [0.6547181060359146, 0.6746765864090367, 0.774…  0.655495
18      (1, 5)  [0.7943717590603673, 0.7591086339758143, 0.779…  0.643808
..         …                                                …        …
3         (3,)  [-3.363320503244287, -4.765335421676004, -2.57…  -4.46804
75   (2, 3, 7)  [-4.392650538227055, -3.832124525089191, -2.53… -4.496315
29      (3, 7)  [-4.356124743352352, -3.760094462772833, -2.61… -4.610686
10      (0, 3)  [-3.034571094035231, -10.435376268817596, -3.0… -4.739993
49   (0, 3, 6)  [-4.367703620791792, -2.9297477962333405, -2.6… -4.924148

   feature_names  ci_bound   std_dev   std_err
59     (1, 2, 5)  0.210573  0.163833  0.081917
44     (0, 2, 5)  0.197554  0.153704  0.076852
66     (1, 4, 5)  0.289121  0.224946  0.112473
76     (2, 4, 5)  0.182072  0.141658  0.070829
18        (1, 5)  0.306638  0.238575  0.119287
..           …         …         …        …
3           (3,)  3.422558  2.662866  1.331433
75     (2, 3, 7)  3.198328  2.488408  1.244204
29        (3, 7)  3.483683  2.710424  1.355212
10        (0, 3)  4.159436  3.236182  1.618091
49     (0, 3, 6)  4.105565  3.194269  1.597135

[92 rows x 7 columns]
```

[88]:
```python
#Transformer le dataset pour garder seulement les attributs sélectionnés
X_train_efs = efs.transform(X_train)
```

```
X_test_efs = efs.transform(X_test)
```

[89]:
```
# Entrainer le modèle de régression sur les données d'entrainements réduites
model_MLP = MLPRegressor(hidden_layer_sizes={100, 200,10}, activation='relu',
solver='adam', max_iter= 5000)
model_MLP.fit(X_train_efs, y_train)
#Prédiction sur les données de test
y_pred = model_MLP.predict(X_test_efs)
#évaluer les predictions
score =r2_score(y_test, y_pred)
print('R2 : %.3f' %score)
score1 = mean_squared_error(y_test, y_pred)
print('MSE',score1)
score2 = mean_absolute_error(y_test, y_pred)
print('MAE',score2)
```

```
R2 : 0.794
MSE 13.141802944204397
MAE 2.4906789019905093
```

[90]:
```
from sklearn.pipeline import Pipeline
clf = Pipeline([
 ('feature_selection', EFS(MLPRegressor(hidden_layer_sizes={100, 200,10},␣
 ↪activation='relu',
solver='adam', max_iter= 5000), scoring='r2',
max_features=6, cv=5)),
 ('classification', MLPRegressor(hidden_layer_sizes={100, 200,10},␣
 ↪activation='relu',
solver='adam', max_iter= 5000))
])
clf.fit(X_train, y_train)
clf.score(X_test, y_test)
```

```
Features: 246/246
```

[90]:
```
0.797317371906408
```