Name : Tazmeen Afroz

Roll : 22P-9252

Section : BA-5A

ML-Lab-Task-05

In [51]:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split, cross_val_score, GridS
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler, FunctionTransformer
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline, make_pipeline
from sklearn.cluster import KMeans
from sklearn.metrics.pairwise import rbf_kernel
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import root_mean_squared_error


url = "https://archive.ics.uci.edu/ml/machine-learning-databases/auto-mpg/au
column_names = ["mpg", "cylinders", "displacement", "horsepower", "weight", 
df = pd.read_csv(url, names=column_names, na_values="?", comment="\t", sep="
df = df.drop("name", axis=1)

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

# Split the data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, rar

# Custom StandardScalerClone
class StandardScalerClone(BaseEstimator, TransformerMixin):
    def __init__(self, with_mean=True):
        self.with_mean = with_mean

    def fit(self, X, y=None):
        self.mean_ = X.mean(axis=0)
        self.scale_ = X.std(axis=0)
        return self

    def transform(self, X):
        X_scaled = X.copy()
        if self.with_mean:
            X_scaled -= self.mean_
        return X_scaled / self.scale_
```

```
# ClusterSimilarity transformer
class ClusterSimilarity(BaseEstimator, TransformerMixin):
    def __init__(self, n_clusters=10, gamma=1., random_state=None):
        self.n_clusters = n_clusters
        self.gamma = gamma
        self.random_state = random_state

    def fit(self, X, y=None):
        self.kmeans_ = KMeans(n_clusters=self.n_clusters, random_state=self.
        self.kmeans_.fit(X)
        return self

    def transform(self, X):
        return rbf_kernel(X, self.kmeans_.cluster_centers_, gamma=self.gamma


def column_ratio(X):
    return X[:, [0]] / X[:, [1]]

def ratio_pipeline(name=None):
    return make_pipeline(
        SimpleImputer(strategy="median"),
        FunctionTransformer(column_ratio, feature_names_out=lambda input_fea
        StandardScalerClone()
    )
# Preprocessing pipelines
log_pipeline = make_pipeline(
    SimpleImputer(strategy="median"),
    FunctionTransformer(np.log),
    StandardScalerClone()
)

default_num_pipeline = make_pipeline(
    SimpleImputer(strategy="median"),
    StandardScalerClone()
)

cat_pipeline = make_pipeline(
    SimpleImputer(strategy="most_frequent")
)

geo_pipeline = make_pipeline(
    SimpleImputer(strategy="median"),
    ClusterSimilarity(n_clusters=10, gamma=1., random_state=42)
)

# Main preprocessing pipeline
preprocessing = ColumnTransformer([
    ("horsepower_weight_ratio", ratio_pipeline("horsepower_weight_ratio"), [
    ("displacement_weight_ratio", ratio_pipeline("displacement_weight_ratio"
    ("log", log_pipeline, ["horsepower", "weight", "displacement", "accelera
    ("geo", geo_pipeline, ["horsepower", "weight"]),  # Use the updated geo 
    ("cat", cat_pipeline, ["cylinders", "model_year", "origin"]),
],
remainder=default_num_pipeline)
```

```
# Linear Regression
lin_reg = make_pipeline(preprocessing, LinearRegression())
lin_reg.fit(X_train, y_train)
auto_predictions = lin_reg.predict(X_train)
print("Linear Regression predictions (first 5):", auto_predictions[:5].rounc
lin_rmse = root_mean_squared_error(y_train, auto_predictions)
print("Linear Regression RMSE:", lin_rmse)
```

```
Linear Regression predictions (first 5): [14.7  25.14 35.61 35.34 23.8 ]
Linear Regression RMSE: 3.062376431024096
```

In [52]:

```
from sklearn.tree import DecisionTreeRegressor
tree_reg = make_pipeline(preprocessing,
DecisionTreeRegressor(random_state=42))
tree_reg.fit(X_train, y_train)
auto_predictions = tree_reg.predict(X_train)
print("Decision Tree predictions (first 5):", auto_predictions[:5].round(2))

tree_rmse = root_mean_squared_error(y_train, auto_predictions)
print("Decision Tree RMSE:", tree_rmse)
```

```
Decision Tree predictions (first 5): [16.  27.  37.  36.1 23. ]
Decision Tree RMSE: 0.0
```

In [53]:

```
from sklearn.model_selection import cross_val_score
tree_rmses = -cross_val_score(tree_reg, X_train, y_train, scoring="neg_root_
pd.Series(tree_rmses).describe()
```

Out[53]:

```
count    10.000000
mean      4.119293
std       0.501744
min       3.440703
25%       3.786239
50%       4.127104
75%       4.320897
max       5.038818
dtype: float64
```

In [54]:

```
from sklearn.ensemble import RandomForestRegressor
forest_reg = make_pipeline(preprocessing,
RandomForestRegressor(random_state=42))
forest_rmses = -cross_val_score(forest_reg, X_train, y_train, scoring="neg_r
pd.Series(forest_rmses).describe()
```

Out[54]:

```
count    10.000000
mean      2.967987
std       0.559933
min       2.108097
25%       2.549907
50%       2.940715
75%       3.350417
max       3.940770
dtype: float64
```

In [55]:

```
print(preprocessing.get_params().keys())
```

```
dict_keys(['force_int_remainder_cols', 'n_jobs', 'remainder__memory', 'remai
nder__steps', 'remainder__verbose', 'remainder__simpleimputer', 'remainder__
standardscalerclone', 'remainder__simpleimputer__add_indicator', 'remainder_
_simpleimputer__copy', 'remainder__simpleimputer__fill_value', 'remainder__s
impleimputer__keep_empty_features', 'remainder__simpleimputer__missing_value
s', 'remainder__simpleimputer__strategy', 'remainder__standardscalerclone__w
ith_mean', 'remainder', 'sparse_threshold', 'transformer_weights', 'transfor
mers', 'verbose', 'verbose_feature_names_out', 'horsepower_weight_ratio', 'd
isplacement_weight_ratio', 'log', 'geo', 'cat', 'horsepower_weight_ratio__me
mory', 'horsepower_weight_ratio__steps', 'horsepower_weight_ratio__verbose',
'horsepower_weight_ratio__simpleimputer', 'horsepower_weight_ratio__function
transformer', 'horsepower_weight_ratio__standardscalerclone', 'horsepower_we
ight_ratio__simpleimputer__add_indicator', 'horsepower_weight_ratio__simplei
mputer__copy', 'horsepower_weight_ratio__simpleimputer__fill_value', 'horsep
ower_weight_ratio__simpleimputer__keep_empty_features', 'horsepower_weight_r
atio__simpleimputer__missing_values', 'horsepower_weight_ratio__simpleimpute
r__strategy', 'horsepower_weight_ratio__functiontransformer__accept_sparse',
'horsepower_weight_ratio__functiontransformer__check_inverse', 'horsepower_w
eight_ratio__functiontransformer__feature_names_out', 'horsepower_weight_rat
io__functiontransformer__func', 'horsepower_weight_ratio__functiontransforme
r__inv_kw_args', 'horsepower_weight_ratio__functiontransformer__inverse_fun
c', 'horsepower_weight_ratio__functiontransformer__kw_args', 'horsepower_wei
ght_ratio__functiontransformer__validate', 'horsepower_weight_ratio__standar
dscalerclone__with_mean', 'displacement_weight_ratio__memory', 'displacement
_weight_ratio__steps', 'displacement_weight_ratio__verbose', 'displacement_w
eight_ratio__simpleimputer', 'displacement_weight_ratio__functiontransforme
r', 'displacement_weight_ratio__standardscalerclone', 'displacement_weight_r
atio__simpleimputer__add_indicator', 'displacement_weight_ratio__simpleimput
er__copy', 'displacement_weight_ratio__simpleimputer__fill_value', 'displace
ment_weight_ratio__simpleimputer__keep_empty_features', 'displacement_weight
_ratio__simpleimputer__missing_values', 'displacement_weight_ratio__simpleim
puter__strategy', 'displacement_weight_ratio__functiontransformer__accept_sp
arse', 'displacement_weight_ratio__functiontransformer__check_inverse', 'dis
placement_weight_ratio__functiontransformer__feature_names_out', 'displaceme
nt_weight_ratio__functiontransformer__func', 'displacement_weight_ratio__fun
ctiontransformer__inv_kw_args', 'displacement_weight_ratio__functiontransfor
mer__inverse_func', 'displacement_weight_ratio__functiontransformer__kw_arg
s', 'displacement_weight_ratio__functiontransformer__validate', 'displacemen
t_weight_ratio__standardscalerclone__with_mean', 'log__memory', 'log__step
s', 'log__verbose', 'log__simpleimputer', 'log__functiontransformer', 'log__
standardscalerclone', 'log__simpleimputer__add_indicator', 'log__simpleimput
er__copy', 'log__simpleimputer__fill_value', 'log__simpleimputer__keep_empty
_features', 'log__simpleimputer__missing_values', 'log__simpleimputer__strat
egy', 'log__functiontransformer__accept_sparse', 'log__functiontransformer__
check_inverse', 'log__functiontransformer__feature_names_out', 'log__functio
ntransformer__func', 'log__functiontransformer__inv_kw_args', 'log__function
transformer__inverse_func', 'log__functiontransformer__kw_args', 'log__funct
iontransformer__validate', 'log__standardscalerclone__with_mean', 'geo__memo
ry', 'geo__steps', 'geo__verbose', 'geo__simpleimputer', 'geo__clustersimila
rity', 'geo__simpleimputer__add_indicator', 'geo__simpleimputer__copy', 'geo
__simpleimputer__fill_value', 'geo__simpleimputer__keep_empty_features', 'ge
o__simpleimputer__missing_values', 'geo__simpleimputer__strategy', 'geo__clu
stersimilarity__gamma', 'geo__clustersimilarity__n_clusters', 'geo__clusters
imilarity__random_state', 'cat__memory', 'cat__steps', 'cat__verbose', 'cat_
_simpleimputer', 'cat__simpleimputer__add_indicator', 'cat__simpleimputer__c
opy', 'cat__simpleimputer__fill_value', 'cat__simpleimputer__keep_empty_feat
ures', 'cat__simpleimputer__missing_values', 'cat__simpleimputer__strateg
y'])
```

In [56]:

```python
from sklearn.model_selection import GridSearchCV


full_pipeline = Pipeline([

("preprocessing", preprocessing),
("random_forest", RandomForestRegressor(random_state=42)),
])

param_grid = [
    {'preprocessing__geo__clustersimilarity__n_clusters': [3,5,7],
     'random_forest__max_features': [4, 6, 8]},
    {'preprocessing__geo__clustersimilarity__n_clusters': [3, 4],
     'random_forest__max_features': [6, 8, 10]},
]

grid_search = GridSearchCV(full_pipeline, param_grid, cv=3,
                           scoring='neg_root_mean_squared_error')
grid_search.fit(X_train, y_train)

print("Best parameters:", grid_search.best_params_)
print("Best RMSE:", -grid_search.best_score_)

# Evaluate the best model on the test set
final_model = grid_search.best_estimator_
final_predictions = final_model.predict(X_test)
final_rmse = root_mean_squared_error(y_test, final_predictions)
print("Final model RMSE on test set:", final_rmse)
```

```
Best parameters: {'preprocessing__geo__clustersimilarity__n_clusters': 3, 'r
andom_forest__max_features': 8}
Best RMSE: 3.042088381949943
Final model RMSE on test set: 2.17829597965933
```

In [57]:

```python
cv_res = pd.DataFrame(grid_search.cv_results_)
cv_res.sort_values(by="mean_test_score", ascending=False,inplace=True)
cv_res.head()
```

Out[57]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_preprocessing_
2	0.165242	0.009191	0.012923	0.002659	
10	0.172137	0.015288	0.014452	0.001053	
12	0.133769	0.002908	0.011041	0.000427	
13	0.137349	0.001288	0.011055	0.000065	
14	0.149653	0.002065	0.012944	0.002220	

In [58]:

```python
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint

param_distribs = {
    'preprocessing__geo__clustersimilarity__n_clusters': randint(low=3, high
    'random_forest__max_features': randint(low=2, high=20)
}

rnd_search = RandomizedSearchCV(
    full_pipeline, param_distributions=param_distribs, n_iter=10,
    cv=3, scoring='neg_root_mean_squared_error', random_state=42
)
rnd_search.fit(X_train, y_train)

print("Best parameters:", rnd_search.best_params_)
print("Best RMSE:", -rnd_search.best_score_)

# Evaluate the best model on the test set
final_model = rnd_search.best_estimator_
final_predictions = final_model.predict(X_test)
final_rmse = root_mean_squared_error(y_test, final_predictions)
print("Final model RMSE on test set:", final_rmse)
```

```
Best parameters: {'preprocessing__geo__clustersimilarity__n_clusters': 41,
'random_forest__max_features': 16}
Best RMSE: 3.058513857671436
Final model RMSE on test set: 2.218441933542547
```