Министерство науки и высшего образования Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования «Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)» (МГТУ им. Н.Э. Баумана)

Работа №4

по курсу «Технологии машинного обучения»

Выполнил

студент группы ИУ5-64Б

Шпак И.Д.

Москва, 2020

1 Исходное задание

1. Выберите набор данных (датасет) для решения задачи классификации или регрессии.
2. В случае необходимости проведите удаление или заполнение пропусков и кодирование категориальных признаков
3. С использованием метода train_test_split разделите выборку на обучающую и тестовую. с использованием GridSearchCV и/или RandomizedSearchCV и кросс-валидации, оцените качество оптимальной модели. Желательно использование нескольких стратегий кросс-валидации.
4. Обучите следующие модели:
   - одну из линейных моделей;
   - SVM;
   - дерево решений.
5. Оцените качество моделей с помощью двух подходящих для задачи метрик. Сравните качество полученных моделей.

2 Код программы

[105]:
```
from IPython.display import Image
import numpy as np
import pandas as pd
from sklearn import svm, datasets
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Ridge
from sklearn.model_selection import cross_val_score, cross_validate
from sklearn.model_selection import KFold, RepeatedKFold, LeaveOneOut, LeavePOut,␣
↪ShuffleSplit, StratifiedKFold
from sklearn.metrics import mean_absolute_error, mean_squared_error,␣
↪mean_squared_log_error, median_absolute_error, r2_score,␣
↪mean_absolute_percentage_error
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
```

```python
from sklearn.linear_model import ElasticNet
from sklearn.svm import LinearSVR,SVR
from sklearn.preprocessing import OrdinalEncoder, StandardScaler
from sklearn.tree import DecisionTreeRegressor
from sklearn.datasets import load_wine
from io import StringIO
%matplotlib inline
pd.set_option("display.max_rows", None, "display.max_columns", None)
sns.set(style="ticks")
```

[106]:
```python
data = pd.read_csv("/home/igor/Downloads/CarPrice_Assignment.xls",sep=',')
data.shape
```

[106]: (205, 26)

[107]:
```python
cleanup_nums = {"doornumber":     {"four": 1, "two": 0},
                "cylindernumber": {"four": 4, "six": 6, "five": 5, "eight": 8,
                                  "two": 2, "twelve": 12, "three":3 },
                "aspiration":{"std": 0, "turbo": 1},
                "fueltype":{"gas": 0, "diesel": 1},
                "enginelocation":{"front": 0, "rear": 1}}
data = data.replace(cleanup_nums)
data=pd.get_dummies(data, columns=["drivewheel"], prefix=["drive"])
data=pd.get_dummies(data, columns=["carbody"], prefix=["body"])
data["OHC_Code"] = np.where(data["enginetype"].str.contains("ohc"), 1, 0)
data.drop(data[(data['aspiration']=='turbo')].index,inplace=True)
data.drop(data[(data['fueltype']=='diesel')].index,inplace=True)
data.
 ↪drop(["CarName","enginetype","fuelsystem","symboling","car_ID"],axis=1,inplace=True)
```

[108]:
```python
data_X = data.loc[:,data.columns]
clnm = StandardScaler()
data_X = clnm.fit_transform(data_X)
data_X = pd.DataFrame(data_X,columns=data.columns)
data_Y = data.loc[:, 'price']
data_X.drop(['price'],axis=1,inplace=True)
data_X_train, data_X_test, data_y_train, data_y_test = train_test_split(
    data_X, data_Y,test_size=0.2, random_state=360)
data_Y.head()
```

[108]:
```
0    13495.0
1    16500.0
2    16500.0
3    13950.0
4    17450.0
Name: price, dtype: float64
```

[109]:
```
data_X = data_X.to_numpy()
data_Y = data_Y.to_numpy()
data_X_train, data_X_test, data_Y_train, data_Y_test = train_test_split(
    data_X, data_Y,test_size=0.2, random_state=360)
```

[110]:
```
data_X_train.shape
```

[110]:
```
(164, 27)
```

[111]:
```
reg = Ridge(alpha = 0.1).fit(data_X_train.reshape(-1, 27), data_Y_train)
reg.coef_, reg.intercept_
target1_0 = reg.predict(data_X_train)
target1_1 = reg.predict(data_X_test)
r2_score(data_Y_test, target1_1), mean_absolute_error(data_Y_test, target1_1)
```

[111]:
```
(0.8866403748933834, 2181.032907732371)
```

[112]:
```
scores = cross_val_score(Ridge(alpha = 1),
                         data_X, data_Y,
                         cv=4)
print("%0.2f r^2 with a standard deviation of %0.2f" % (scores.mean(), scores.
 ↪std()))
```

```
0.34 r^2 with a standard deviation of 0.46
```

[113]:
```
a =np.linspace(0.01,1,100)
grid = GridSearchCV(estimator = Ridge() ,param_grid={'alpha': ␣
 ↪a},cv=RepeatedKFold(n_splits=3, n_repeats=3),scoring="r2")
grid.fit(data_X,data_Y)
grid.best_score_ , grid.best_params_,grid.best_estimator_
```

[113]:
```
(0.8481441870301406, {'alpha': 1.0}, Ridge())
```

[114]:
```
grid.best_estimator_.fit(data_X_train, data_Y_train)
target2_0 = grid.best_estimator_.predict(data_X_train)
target2_1 = grid.best_estimator_.predict(data_X_test)
```

```
r2_score(data_Y_test, target2_1), mean_absolute_error(data_Y_test, target2_1)
```

[114]: (0.9102807327732504, 1937.558854553965)

[115]:
```
scores = cross_val_score(grid.best_estimator_, data_X, data_Y,␣
↪cv=RepeatedKFold(n_splits=3, n_repeats=3))
print("%0.2f r^2 with a standard deviation of %0.2f" % (scores.mean(), scores.
↪std()))
```

```
0.82 r^2 with a standard deviation of 0.03
```

[116]:
```
grid = GridSearchCV(estimator = Lasso(tol=1e-1) ,param_grid={'alpha': ␣
↪a},cv=RepeatedKFold(n_splits=3, n_repeats=3),scoring="r2")
grid.fit(data_X,data_Y)
grid.best_score_ , grid.best_params_,grid.best_estimator_
```

[116]: (0.8011735720066059, {'alpha': 1.0}, Lasso(tol=0.1))

[117]:
```
grid.best_estimator_.fit(data_X_train, data_Y_train)
target3_0 = grid.best_estimator_.predict(data_X_train)
target3_1 = grid.best_estimator_.predict(data_X_test)
r2_score(data_Y_test, target3_1),1  mean_absolute_error(data_Y_test, target3_1)
```

[117]: (0.9231639328357497, 1792.2836196899398)

[118]:
```
scores = cross_val_score(grid.best_estimator_, data_X, data_Y,␣
↪cv=RepeatedKFold(n_splits=3, n_repeats=3))
print("%0.2f r^2 with a standard deviation of %0.2f" % (scores.mean(), scores.
↪std()))
```

```
0.77 r^2 with a standard deviation of 0.12
```

[119]:
```
b =np.linspace(0.1,1,10)
grid = GridSearchCV(estimator = ElasticNet(tol=1e-1) ,param_grid={'alpha': ␣
↪a,'l1_ratio' : b},cv=RepeatedKFold(n_splits=3, n_repeats=3),scoring="r2")
grid.fit(data_X,data_Y)
grid.best_score_ , grid.best_params_,grid.best_estimator_
```

[119]:
```
(0.8199309484931719,
 {'alpha': 0.8200000000000001, 'l1_ratio': 0.8},
 ElasticNet(alpha=0.8200000000000001, l1_ratio=0.8, tol=0.1))
```

[120]:
```
grid.best_estimator_.fit(data_X_train, data_Y_train)
target4_0 = grid.best_estimator_.predict(data_X_train)
target4_1 = grid.best_estimator_.predict(data_X_test)
r2_score(data_Y_test, target4_1),mean_absolute_error(data_Y_test, target4_1)
```

[120]:
```
(0.9270503695691327, 1758.5738563522843)
```

[121]:
```
scores = cross_val_score(grid.best_estimator_, data_X, data_Y,␣
↪cv=RepeatedKFold(n_splits=3, n_repeats=3))
print("%0.2f r^2 with a standard deviation of %0.2f" % (scores.mean(), scores.
↪std()))
```

```
0.81 r^2 with a standard deviation of 0.04
```

[122]:
```
poly_model = Pipeline([('poly', PolynomialFeatures(degree=3)),
                        ('linear', LinearRegression(fit_intercept=False))])
grid = GridSearchCV(estimator = poly_model ,param_grid={'poly__degree': ␣
↪range(1,5,1)},cv=RepeatedKFold(n_splits=3, n_repeats=3),scoring="r2")
grid.fit(data_X,data_Y)
grid.best_score_ , grid.best_params_,grid.best_estimator_
```

[122]:
```
(0.8239093877500444,
 {'poly__degree': 1},
 Pipeline(steps=[('poly', PolynomialFeatures(degree=1)),
                 ('linear', LinearRegression(fit_intercept=False))]))
```

[123]:
```
grid.best_estimator_.fit(data_X_train, data_Y_train)
target5_0 = grid.best_estimator_.predict(data_X_train)
target5_1 = grid.best_estimator_.predict(data_X_test)
r2_score(data_Y_test, target5_1), mean_absolute_error(data_y_test, target5_1)
```

[123]:
```
(0.8817678394823523, 2225.7013621863985)
```

[124]:
```
scores = cross_val_score(grid.best_estimator_, data_X, data_Y,␣
↪cv=RepeatedKFold(n_splits=3, n_repeats=3))
print("%0.2f r^2 with a standard deviation of %0.2f" % (scores.mean(), scores.
↪std()))
```

```
0.81 r^2 with a standard deviation of 0.03
```

[150]:
```
reg1 = LinearSVR(C=1.0, loss='squared_epsilon_insensitive', max_iter=1000)
reg1.fit(data_X_train, data_Y_train)
target6_0=reg1.predict(data_X_train)
```

```
target6_1=reg1.predict(data_X_test)
r2_score(data_Y_test, target6_1), mean_absolute_error(data_y_test, target6_1)
```

[150]: (0.8994700594977598, 2049.1038225360794)

[126]:
```
reg2 = DecisionTreeRegressor(random_state=360,max_depth=4)
reg2.fit(data_X_train, data_Y_train)
target7_0=reg2.predict(data_X_test)
sum(reg2.feature_importances_)
```

[126]: 1.0

[127]:
```
r2_score(data_Y_test, target7_0),mean_absolute_error(data_Y_test, target7_0)
```

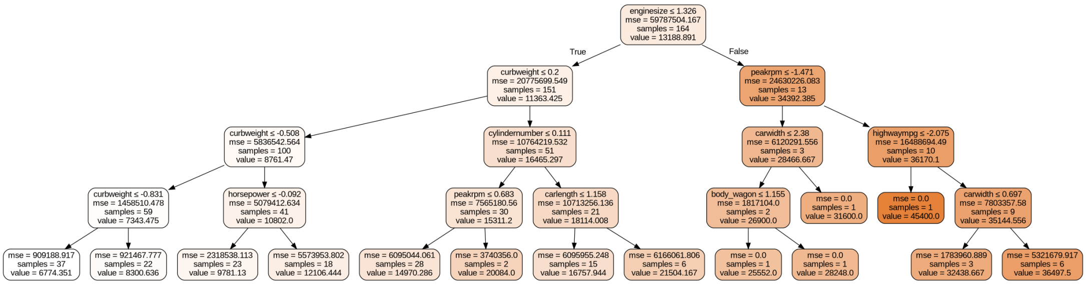
[127]: (0.9106421039901045, 1899.044949927358)

[128]:
```
scores = cross_val_score(reg2, data_X, data_Y, cv=RepeatedKFold(n_splits=3,␣
↪n_repeats=3))
print("%0.2f r^2 with a standard deviation of %0.2f" % (scores.mean(), scores.
↪std()))
```

0.85 r^2 with a standard deviation of 0.05

[129]:
```
#
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor,␣
↪export_graphviz
import graphviz
import pydotplus
data_X = data.loc[:,data.columns]
data_X.drop(['price'],axis=1,inplace=True)
def get_png_tree(tree_model_param, feature_names_param):
    dot_data = StringIO()
    export_graphviz(tree_model_param, out_file=dot_data,␣
↪feature_names=feature_names_param,
                    filled=True, rounded=True, special_characters=True)
    graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
    return graph.create_png()
Image(get_png_tree(reg2, data_X.columns), height='70%')
```

[129]:



enginesize ≤ 1.326
mse = 59787504.167
samples = 164
value = 13188.891
True
False
curbweight ≤ 0.2
mse = 20775699.549
samples = 151
value = 11363.425
peakrpm ≤ -1.471
mse = 24630226.083
samples = 13
value = 34392.385
curbweight ≤ -0.508
mse = 5836542.564
samples = 100
value = 8761.47
cylindernumber ≤ 0.111
mse = 10764219.532
samples = 51
value = 16465.297
carwidth ≤ 2.38
mse = 6120291.556
samples = 3
value = 28466.667
highwaympg ≤ -2.075
mse = 16488694.49
samples = 10
value = 36170.1
curbweight ≤ -0.831
mse = 1458510.478
samples = 59
value = 7343.475
horsepower ≤ -0.092
mse = 5079412.634
samples = 41
value = 10802.0
peakrpm ≤ 0.683
mse = 7565180.56
samples = 30
value = 15311.2
carlength ≤ 1.158
mse = 10713256.136
samples = 21
value = 18114.008
body_wagon ≤ 1.155
mse = 1817104.0
samples = 2
value = 26900.0
mse = 0.0
samples = 1
value = 31600.0
mse = 0.0
samples = 1
value = 45400.0
carwidth ≤ 0.697
mse = 7803357.58
samples = 9
value = 35144.556
mse = 909188.917
samples = 37
value = 6774.351
mse = 921467.777
samples = 22
value = 8300.636
mse = 2318538.113
samples = 23
value = 9781.13
mse = 5573953.802
samples = 18
value = 12106.444
mse = 6095044.061
samples = 28
value = 14970.286
mse = 3740356.0
samples = 2
value = 20084.0
mse = 6095955.248
samples = 15
value = 16757.944
mse = 6166061.806
samples = 6
value = 21504.167
mse = 0.0
samples = 1
value = 25552.0
mse = 0.0
samples = 1
value = 28248.0
mse = 1783960.889
samples = 3
value = 32438.667
mse = 5321679.917
samples = 6
value = 36497.5