

МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ  
им. Н.Э. Баумана

Факультет «Информатика и системы управления»  
Кафедра «Систем обработки информации и управления»

ОТЧЕТ

**Лабораторная работа №5**  
по дисциплине «Методы машинного обучения»

Тема: «Линейные модели, SVM и деревья решений»

ИСПОЛНИТЕЛЬ:  
группа ИУ5-22М

\_\_\_ Паршева Анна \_\_\_  
ФИО

\_\_\_\_\_  
подпись

" \_\_ " \_\_\_\_\_ 2020 г.

Москва - 2020

---

# 1. Задание

- Выберите набор данных (датасет) для решения задачи классификации или регрессии.
- В случае необходимости проведите удаление или заполнение пропусков и кодирование категориальных признаков.
- С использованием метода `train_test_split` разделите выборку на обучающую и тестовую.
- Обучите следующие модели: одну из линейных моделей; SVM; дерево решений.
- Оцените качество моделей с помощью трех подходящих для задачи метрик. Сравните качество полученных моделей.
- Произведите для каждой модели подбор одного гиперпараметра с использованием `GridSearchCV` и кросс-валидации.
- Повторите пункт 4 для найденных оптимальных значений гиперпараметров. Сравните качество полученных моделей с качеством моделей, полученных в пункте 4.

## 2. Подготовка данных

Выбранный набор данных представляет из себя набор характеристик различных автомобилей, целевой переменной является - цена автомобиля

```
In [12]: import pandas as pd

df = pd.read_csv('CarPrice.csv')
df.head()
```

Out[12]:

	car_ID	symboling	CarName	fueltype	aspiration	doornumber	carbody	drivewheel
0	1	3	alfa-romero giulia	gas	std	two	convertible	rwd
1	2	3	alfa-romero stelvio	gas	std	two	convertible	rwd
2	3	1	alfa-romero Quadrifoglio	gas	std	two	hatchback	rwd
3	4	2	audi 100 ls	gas	std	four	sedan	fwd
4	5	2	audi 100ls	gas	std	four	sedan	4wd

5 rows × 26 columns

```
In [14]: row_number = df.shape[0]
column_number = df.shape[1]

print('Данный датасет содержит {} строк и {} столбцов.'.format(row_
number, column_number))
```

Данный датасет содержит 205 строк и 26 столбцов.

```
In [15]: null_flag = False
null_columns = {}
for col in df.columns:
    null_count = df[df[col].isnull()].shape[0]
    if null_count > 0:
        null_flag = True
        column_type = df[col].dtype
        null_columns[col] = column_type
        percent = round((null_count / row_number) * 100, 3)
        print('{} - {} - {}. Тип - {}'.format(col, null_count, perc
ent, column_type))

if not null_flag:
    print('Пропуски в данных отсутствуют.')
```

Пропуски в данных отсутствуют.

```
In [17]: for col in df.columns:
column_type = df[col].dtype
if column_type == 'object':
    print(col)
```

CarName  
fueltype  
aspiration  
doornumber  
carbody  
drivewheel  
enginelocation  
enginetype  
cylindernumber  
fuelsystem

```
In [18]: from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()

for col in df.columns:
    df[col] = le.fit_transform(df[col])
```

```
In [19]: for col in df.columns:
          column_type = df[col].dtype
          if column_type == 'object':
              print(col)
```

```
In [73]: from sklearn.model_selection import train_test_split

df_x = df.loc[:, df.columns != 'price']
df_y = df['price']
train_x_df, test_x_df, train_y_df, test_y_df = train_test_split(df_x, df_y,
                                                                test_size=0.3, random_state=1)
```

```
In [43]: row_number_train = train_x_df.shape[0]
          column_number_train = train_x_df.shape[1]

          print('Тренировочный датасет содержит {} строки и {} столбцов.'.format(row_number_train, column_number_train))
```

Тренировочный датасет содержит 143 строки и 25 столбцов.

```
In [42]: row_number_test = test_x_df.shape[0]
          column_number_test = test_x_df.shape[1]

          print('Тестовый датасет содержит {} строки и {} столбцов.'.format(row_number_test, column_number_test))
```

Тестовый датасет содержит 62 строки и 25 столбцов.

## 3. Обучение моделей

### 3.1 Линейная модель

#### 3.1.1 Матрица корреляции

```
In [38]: corr_matrix = df.corr()
          corr_matrix
```

Out[38]:

	car_ID	symboling	CarName	fueltype	aspiration	doornumber	carbody
<b>car_ID</b>	1.000000	-0.151621	0.967077	-0.125568	0.067729	-0.190352	0.098303
<b>symboling</b>	-0.151621	1.000000	-0.107095	0.194311	-0.059866	0.664073	-0.596135
<b>CarName</b>	0.967077	-0.107095	1.000000	-0.069435	0.019914	-0.171745	0.099691
<b>fueltype</b>	-0.125568	0.194311	-0.069435	1.000000	-0.401397	0.191491	-0.147853
<b>aspiration</b>	0.067729	-0.059866	0.019914	-0.401397	1.000000	-0.031792	0.063028
<b>doornumber</b>	-0.190352	0.664073	-0.171745	0.191491	-0.031792	1.000000	-0.680358
<b>carbody</b>	0.098303	-0.596135	0.099691	-0.147853	0.063028	-0.680358	1.000000
<b>drivewheel</b>	0.051406	-0.041671	-0.016129	-0.132257	0.066465	0.098954	-0.119667
<b>enginelocation</b>	0.051483	0.212471	0.055968	0.040070	-0.057191	0.137757	-0.208850
<b>wheelbase</b>	0.162792	-0.535721	0.051429	-0.296072	0.246290	-0.466657	0.007118
<b>carlength</b>	0.162165	-0.357163	0.051896	-0.196281	0.244096	-0.382903	-0.201092
<b>carwidth</b>	0.104421	-0.234801	-0.010630	-0.231914	0.311695	-0.229790	0.345925
<b>carheight</b>	0.255711	-0.516952	0.194678	-0.299030	0.121833	-0.550000	-0.211981
<b>curbweight</b>	0.119667	-0.208850	0.007118	-0.201092	0.345925	-0.211981	0.119667
<b>enginetype</b>	-0.075130	0.050372	-0.090381	0.082695	-0.102963	0.062431	-0.075130
<b>cylindernumber</b>	-0.040912	0.197762	0.047154	0.110617	-0.133119	0.154322	-0.040912
<b>enginesize</b>	0.028107	-0.102395	-0.090497	-0.126387	0.197266	-0.036787	-0.102395
<b>fuelsystem</b>	0.204898	0.091163	0.123845	0.041529	0.288086	0.015519	-0.091163
<b>boreratio</b>	0.271661	-0.129044	0.201561	-0.043657	0.210910	-0.117787	0.271661
<b>stroke</b>	-0.175206	-0.018198	-0.205093	-0.302867	0.223521	0.004591	-0.175206
<b>compressionratio</b>	0.175895	-0.053801	0.168305	-0.634479	-0.143321	-0.058337	0.175895
<b>horsepower</b>	-0.003477	0.054142	-0.095958	0.160982	0.307844	0.092022	-0.003477
<b>peakrpm</b>	-0.217333	0.275404	-0.149881	0.487600	-0.181173	0.239094	-0.217333
<b>citympg</b>	0.033055	-0.049180	0.111351	-0.257728	-0.204284	-0.005580	0.033055
<b>highwaympg</b>	0.022503	0.019534	0.116787	-0.173238	-0.264490	0.021068	-0.019534
<b>price</b>	0.012452	-0.089579	-0.096100	-0.140119	0.307676	-0.115160	0.012452

26 rows × 26 columns

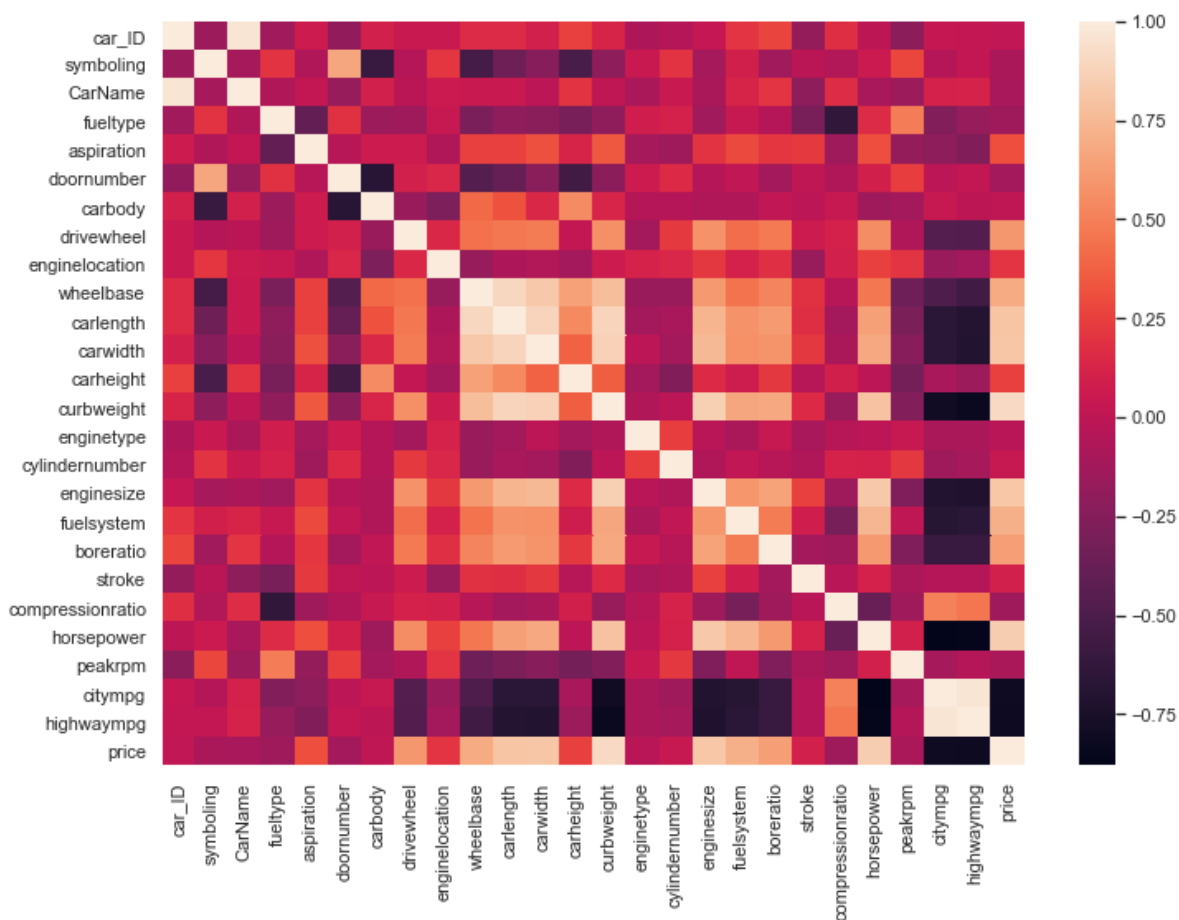
```
In [37]: corr_matrix['price'].nlargest(4)
```

```
Out[37]: price          1.000000  
curbweight    0.905891  
horsepower    0.846130  
enginesize    0.818392  
Name: price, dtype: float64
```

```
In [27]: import seaborn as sns
```

```
sns.set(rc={'figure.figsize':(11.7,8.27)})  
sns.heatmap(corr_matrix)
```

```
Out[27]: <matplotlib.axes._subplots.AxesSubplot at 0x1229205c0>
```



```
In [32]: import plotly.express as px  
fig = px.scatter(df, x='price', y='curbweight')  
fig.show()
```

```
In [33]: fig = px.scatter(df, x='price', y='horsepower')  
fig.show()
```



```
In [39]: fig = px.scatter(df, x='price', y='enginesize')  
fig.show()
```

### 3.1.2 Вычисление коэффициентов регрессии матричным способом

$$B = (X^T X)^{-1} X^T Y$$

```
In [40]: import numpy as np
```

```
In [110]: X_0 = np.ones(row_number_train).T
X = np.column_stack((X_0, train_x_df))
B = np.dot(np.dot(np.linalg.inv(np.dot(X.T,X)),X.T),train_y_df)
B
```

```
Out[110]: array([ 5.24106398e+01, -1.02311681e-01,  2.18470564e+00, -2.11642
101e-02,
                -2.92692816e+01, -5.83570818e+00, -7.93128553e+00, -4.40863
223e+00,
                2.52699519e+00,  3.20434750e+01,  6.34225165e-01, -2.06578
965e-01,
                7.36209558e-01,  2.50456405e-01,  3.60955023e-01,  9.91672
740e-01,
                4.48711749e-01, -4.77936652e-01,  3.58618089e+00, -2.09333
506e-01,
                -5.03714418e-01,  5.71833312e-01,  1.43046552e+00, -1.12281
838e-01,
                -2.40925647e+00,  1.50738855e+00])
```

### 3.1.3 Использование класса LinearRegression библиотеки scikit-learn

#### 3.1.3.1 Обучение с произвольным гиперпараметром

```
In [194]: from sklearn.linear_model import Lasso
reg1 = Lasso(alpha=0.1).fit(np.array(train_x_df),
                             np.array(train_y_df).reshape(-1, 1))
B_1 = (reg1.intercept_, reg1.coef_)
```

```
In [195]: B_1
```

```
Out[195]: (array([42.14887115]),
array([-1.11567509e-01,  1.84913331e+00, -1.05906495e-02, -2.0189
2043e+01,
        -6.63162812e-01, -7.31529743e+00, -4.77816533e+00,  1.8302
3531e+00,
        2.38949769e+01,  6.35998365e-01, -2.13153098e-01,  6.5925
4648e-01,
        2.64508407e-01,  3.75002489e-01,  9.35438073e-01,  4.1329
2452e-01,
        -2.53813867e-01,  3.67533788e+00, -1.76396306e-01, -4.9812
8698e-01,
        8.44526642e-01,  1.25702856e+00, -1.09337548e-02, -2.5537
7484e+00,
        1.55622583e+00]))
```

### 3.1.3.2 Оценка качества модели

```
In [196]: predicted_y_reg = reg1.predict(np.array(test_x_df))
```

```
In [197]: predict_test_df = pd.DataFrame(test_y_df)
predict_test_df['predicted_y'] = predicted_y
```

```
In [198]: import plotly.graph_objects as go

fig = go.Figure()
fig.add_trace(go.Scatter(x=np.arange(predict_test_df['predicted_y']
                                     .shape[0]),
                        y=predict_test_df['predicted_y'],
                        name='predicted'
                        ))

fig.add_trace(go.Scatter(x=np.arange(predict_test_df['predicted_y']
                                     .shape[0]),
                        y=predict_test_df['price'],
                        name='test'
                        ))

fig.show()
```

```
In [199]: from sklearn.metrics import r2_score  
  
r2_reg = round(r2_score(test_y_df, predicted_y_reg), 2)  
  
print('Коэффициент детерминации - %.2f' % r2_reg)
```

Коэффициент детерминации - 0.88

### 3.1.3.3 Подбор гиперпараметра

```

In [209]: from sklearn.model_selection import GridSearchCV
          from sklearn.model_selection import RepeatedKFold

          n_range = np.arange(0.1, 1.2, 0.1)
          tuned_parameters = [{'alpha': n_range}]

          gs = GridSearchCV(Lasso(),
                           param_grid=tuned_parameters,
                           cv=RepeatedKFold(n_splits=3, n_repeats=2),
                           scoring='r2')
          gs.fit(train_x_df, train_y_df)

Out[209]: GridSearchCV(cv=RepeatedKFold(n_repeats=2, n_splits=3, random_state=None),
                      error_score=nan,
                      estimator=Lasso(alpha=1.0, copy_X=True, fit_intercept=True,
                                     max_iter=1000, normalize=False, positive=False,
                                     precompute=False, random_state=None, selection='cyclic', tol=0.0001, warm_start=False),
                      iid='deprecated', n_jobs=None,
                      param_grid=[{'alpha': array([0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1. , 1.1])}],
                      pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
                      scoring='r2', verbose=0)

```

### 3.1.3.4 Обучение с наилучшим гиперпараметром

```

In [210]: gs.best_estimator_.fit(train_x_df, train_y_df)
          predicted_y_best_reg = gs.best_estimator_.predict(test_x_df)

In [211]: r2_reg_best = round(r2_score(test_y_df, predicted_y_best_reg), 2)

          print('Коэффициент детерминации при случайном гиперпараметре - %.2f'
                % r2_reg)
          print('Коэффициент детерминации при наилучшем гиперпараметре - %.2f'
                % r2_reg_best)

```

Коэффициент детерминации при случайном гиперпараметре - 0.88  
 Коэффициент детерминации при наилучшем гиперпараметре - 0.89

## 3.2 SVM

### 3.2.1 Обучение с произвольным гиперпараметром

```
In [156]: from sklearn.svm import SVR

svr = SVR(kernel='linear', C=100)

svr.fit(train_x_df, train_y_df)

predicted_y_svr = svr.predict(test_x_df)
```

### 3.2.2 Оценка качества модели

```
In [162]: r2 = round(r2_score(test_y_df, predicted_y_svr), 2)

print('Коэффициент детерминации - %.2f' % r2)

Коэффициент детерминации - 0.87
```

### 3.2.3 Подбор гиперпараметра

```
In [158]: from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RepeatedKFold

n_range = np.arange(0.1,1,0.1)
tuned_parameters = [{'C': n_range}]

gs = GridSearchCV(SVR(kernel='linear'),
                  param_grid=tuned_parameters,
                  cv=RepeatedKFold(n_splits=3, n_repeats=2),
                  scoring='r2')
gs.fit(train_x_df, train_y_df)

Out[158]: GridSearchCV(cv=RepeatedKFold(n_repeats=2, n_splits=3, random_state=None),
                      error_score=nan,
                      estimator=SVR(C=1.0, cache_size=200, coef0=0.0, degree=3,
                                     epsilon=0.1, gamma='scale', kernel='linear',
                                     max_iter=-1, shrinking=True, tol=0.001,
                                     verbose=False),
                      iid='deprecated', n_jobs=None,
                      param_grid=[{'C': array([0.1, 0.2, 0.3, 0.4, 0.5, 0.6,
                                                0.7, 0.8, 0.9])}],
                      pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
                      scoring='r2', verbose=0)
```

```
In [159]: 'Наилучшее значение параметра c - ' + str(gs.best_params_['C'])
```

```
Out[159]: 'Наилучшее значение параметра c - 0.9'
```

### 3.2.4 Обучение с наилучшим гиперпараметром

```
In [160]: gs.best_estimator_.fit(train_x_df, train_y_df)
predicted_y_best_svr = gs.best_estimator_.predict(test_x_df)
```

```
In [164]: r2_best = round(r2_score(test_y_df, predicted_y_best_svr), 2)

print('Коэффициент детерминации при случайном гиперпараметре - %.2f' % r2)
print('Коэффициент детерминации при наилучшем гиперпараметре - %.2f' % r2_best)
```

Коэффициент детерминации при случайном гиперпараметре - 0.87  
Коэффициент детерминации при наилучшем гиперпараметре - 0.88

## 3.3 Дерево решений

### 3.3.1 Обучение с произвольным гиперпараметром

```
In [166]: from sklearn.tree import DecisionTreeRegressor

dtr = DecisionTreeRegressor(random_state=1).fit(train_x_df,
                                                train_y_df)

predict_y_dtr = dtr.predict(test_x_df)
```

### 3.3.2 Оценка качества модели

```
In [167]: r2_dtr = round(r2_score(test_y_df, predict_y_dtr), 2)

print('Коэффициент детерминации при случайном гиперпараметре - %.2f'
      % r2_dtr)
```

Коэффициент детерминации при случайном гиперпараметре - 0.80

### 3.3.3 Подбор гиперпараметра



```
In [168]: params = {
    'max_depth': [3, 4, 5, 6],
    'min_samples_leaf': [0.04, 0.06, 0.08],
    'max_features': [0.2, 0.4, 0.6, 0.8]
}

grid = GridSearchCV(estimator=DecisionTreeRegressor(random_state=1)
,
                    param_grid=params, scoring='r2', cv=3, n_jobs=-1)

grid.fit(train_x_df, train_y_df)
```

```
Out[168]: GridSearchCV(cv=3, error_score=nan,
                        estimator=DecisionTreeRegressor(ccp_alpha=0.0, criterion='mse',
                                                         max_depth=None, max_features=None,
                                                         max_leaf_nodes=None,
                                                         min_impurity_decrease=0.0,
                                                         min_impurity_split=None,
                                                         min_samples_leaf=1,
                                                         min_samples_split=2,
                                                         min_weight_fraction_leaf=0.0,
                                                         presort='deprecated',
                                                         random_state=1,
                                                         splitter='best'),
                        iid='deprecated', n_jobs=-1,
                        param_grid={'max_depth': [3, 4, 5, 6],
                                   'max_features': [0.2, 0.4, 0.6, 0.8],
                                   'min_samples_leaf': [0.04, 0.06, 0.08]},
                        pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
                        scoring='r2', verbose=0)
```

```
In [170]: for param in params.keys():
    print('Наилучшее значение параметра %s - ' % param + str(grid.best_params_[param]))
```

```
Наилучшее значение параметра max_depth - 6
Наилучшее значение параметра min_samples_leaf - 0.04
Наилучшее значение параметра max_features - 0.2
```

### 3.2.4 Обучение с наилучшим гиперпараметром

```
In [171]: grid.best_estimator_.fit(train_x_df, train_y_df)
          predicted_y_best_dtr = grid.best_estimator_.predict(test_x_df)
```

```
In [172]: r2_best_dtr = round(r2_score(test_y_df, predicted_y_best_svr), 2)

          print('Коэффициент детерминации при случайном гиперпараметре - %.2f'
                %
                r2_dtr)
          print('Коэффициент детерминации при наилучшем гиперпараметре - %.2f'
                %
                r2_best_dtr)
```

Коэффициент детерминации при случайном гиперпараметре - 0.80

Коэффициент детерминации при наилучшем гиперпараметре - 0.88