Рубежный контроль №2

Группа: ИУ5Ц-83Б

Номер варианта 26

Студент: Аброчнов Егор Сергеевич

Задание:

Для заданного набора данных (по Вашему варианту) постройте модели классификации или регрессии (в зависимости от конкретной задачи, рассматриваемой в наборе данных). Для построения моделей используйте методы 1 и 2 (по варианту для Вашей группы). Оцените качество моделей на основе подходящих метрик качества (не менее двух метрик). Какие метрики качества Вы использовали и почему? Какие выводы Вы можете сделать о качестве построенных моделей? Для построения моделей необходимо выполнить требуемую предобработку данных: заполнение пропусков, кодирование категориальных признаков, и т.д.

Набор данных:

Для сравнения всех моделей ML Его можно использовать для прогнозирования

- Классификация типов звезд: 1) Температура - K 2) L - L/Lo 3) R - R/Ro 4) AM -- Mv 5) Цвет - Общий цвет спектра 6) Спектральный класс - O, B, A, F, G, K,МАССА M/C - https://en.wikipedia.org/wiki/Asteroid_spectral_types 7) Тип - Красный карлик, Коричневый карлик, Белый карлик, Главная последовательность, Супергиганты, Гипергиганты
- Цель:

Тип: от 0 до 5: 1) Красный карлик - 0 2) Коричневый карлик - 1 3) Белый карлик - 2 4) Основная последовательность - 3 5) Супергиганты - 4 6) Гипергиганты - 5

- Математика:

Lo = 3,828 x 10^26 Вт (Средняя яркость Солнца)

Ro = 6,9551 x 10^8 м (Средний радиус Солнца)

Импорт библиотек

```
import numpy as np
import pandas as pd
```

```
import seaborn as sns
import matplotlib
import matplotlib_inline
import matplotlib.pyplot as plt
from IPython.display import Image
from io import StringIO
import graphviz
import pydotplus
from sklearn.model_selection import train_test_split
%matplotlib inline
%matplotlib inline
sns.set(style="ticks")
from IPython.display import set_matplotlib_formats
matplotlib_inline.backend_inline.set_matplotlib_formats("retina")
```

Загрузка данных

```
data = pd.read_csv('Stars.csv', sep=",")
```

Основные характеристики датасета

```
data.head()
```

	Temperature	L	R	A_M	Color	Spectral_Class	Type
0	3068	0.002400	0.1700	16.12	Red	M	0
1	3042	0.000500	0.1542	16.60	Red	M	0
2	2600	0.000300	0.1020	18.70	Red	M	0
3	2800	0.000200	0.1600	16.65	Red	M	0
4	1939	0.000138	0.1030	20.06	Red	M	0

```
# Выведем размер датасета - по итогу получилось:
total_count = data.shape[0]
print('Всего строк: {}'.format(total_count))
total_count = data.shape[1]
print('Всего колонок: {}'.format(total_count))
```

```
Всего строк: 240
Всего колонок: 7
```

```
# Выведем список колонок с их типами.
data.dtypes
```

```
Temperature         int64
L                 float64
R                 float64
A_M               float64
Color              object
Spectral_Class     object
Type                int64
dtype: object
```

```
# Проверил количество пустых значений по колонкам.
for col_empty in data.columns:
```

```
    empty_count = data[data[col_empty].isnull()].shape[0]
    print('{} - {}'.format(col_empty, empty_count))
```

```
Temperature - 0
L - 0
R - 0
A_M - 0
Color - 0
Spectral_Class - 0
Type - 0
```

Количество пустых значений означает, что все значения по этим колонкам заполнены.

Кодирование категориальных признаков

Преобразуем цвета и спектральные классы в числовые значения (label encoding)

```
from sklearn.preprocessing import LabelEncoder, OneHotEncoder

# Создаем новый фрейм данных, содержащий только столбцы типа object
obj_data = data.select_dtypes(include=['object']).copy()

obj_data.head()
```

```
  Color Spectral_Class
0   Red              M
1   Red              M
2   Red              M
3   Red              M
4   Red              M
```

```
data["Spectral_Class"].value_counts()
```

```
M    111
B     46
O     40
A     19
F     17
K      6
G      1
Name: Spectral_Class, dtype: int64
```

```
data["Color"].value_counts()
```

```
Red                   112
Blue                   56
Blue-white             26
Blue White             10
yellow-white            8
White                   7
```

```
Blue white              4
white                   3
Yellowish White         3
yellowish               2
Whitish                 2
Orange                  2
White-Yellow            1
Pale yellow orange      1
Yellowish               1
Orange-Red              1
Blue-White              1
Name: Color, dtype: int64
```

```
data["Color"] = data["Color"].astype('category')
data["Spectral_Class"] = data["Spectral_Class"].astype('category')
```

```
data.dtypes
```

```
Temperature          int64
L                  float64
R                  float64
A_M                float64
Color             category
Spectral_Class    category
Type                 int64
dtype: object
```

```
data["Color_cat"] = data["Color"].cat.codes
data["Spectral_Class_cat"] = data["Spectral_Class"].cat.codes
data.head()
```

```
   Temperature         L       R    A_M Color Spectral_Class  Type
Color_cat  \
0         3068  0.002400  0.1700  16.12   Red              M     0
8
1         3042  0.000500  0.1542  16.60   Red              M     0
8
2         2600  0.000300  0.1020  18.70   Red              M     0
8
3         2800  0.000200  0.1600  16.65   Red              M     0
8
4         1939  0.000138  0.1030  20.06   Red              M     0
8

   Spectral_Class_cat
0                   5
1                   5
2                   5
3                   5
4                   5
```

```
data = data.drop(columns='Color')
data = data.drop(columns='Spectral_Class')
```

```
data.head()
```

```
   Temperature         L       R    A_M  Type  Color_cat
Spectral_Class_cat
0         3068  0.002400  0.1700  16.12     0          8
5
1         3042  0.000500  0.1542  16.60     0          8
5
2         2600  0.000300  0.1020  18.70     0          8
5
3         2800  0.000200  0.1600  16.65     0          8
5
4         1939  0.000138  0.1030  20.06     0          8
5
```

```
data.dtypes
```

```
Temperature             int64
L                     float64
R                     float64
A_M                   float64
Type                    int64
Color_cat                int8
Spectral_Class_cat       int8
dtype: object
```

Масштабирование данных

```
from sklearn.preprocessing import MinMaxScaler
```

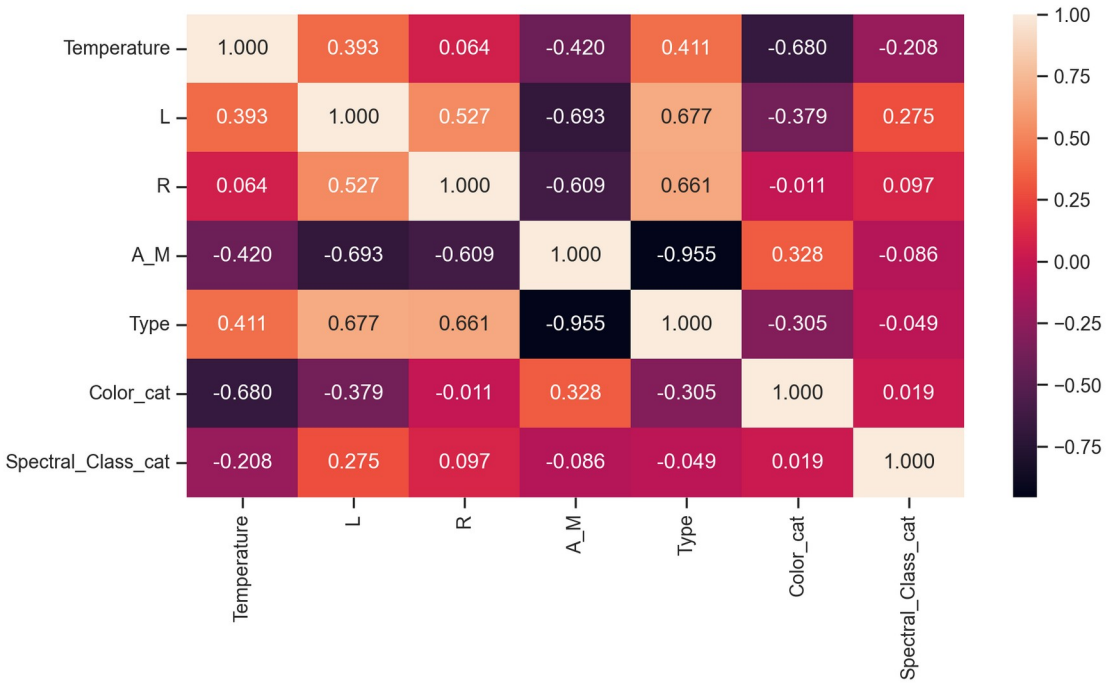
```
sc1 = MinMaxScaler()
sc1_data = sc1.fit_transform(data)
sc1_data
```

```
array([[2.96629095e-02, 2.73127546e-09, 8.29359490e-05, ...,
        0.00000000e+00, 5.00000000e-01, 8.33333333e-01],
       [2.89797956e-02, 4.94455040e-10, 7.48271124e-05, ...,
        0.00000000e+00, 5.00000000e-01, 8.33333333e-01],
       [1.73668585e-02, 2.59000259e-10, 4.80371586e-05, ...,
        0.00000000e+00, 5.00000000e-01, 8.33333333e-01],
       ...,
       [1.81025196e-01, 6.32776483e-01, 7.30304200e-01, ...,
        1.00000000e+00, 5.62500000e-01, 0.00000000e+00],
       [1.91692283e-01, 4.76725295e-01, 5.70693556e-01, ...,
        1.00000000e+00, 5.62500000e-01, 0.00000000e+00],
       [9.44352487e-01, 3.47181606e-01, 9.15062503e-01, ...,
        1.00000000e+00, 0.00000000e+00, 1.00000000e+00]])
```

Построим кореляционную матрицу

```
ig, ax = plt.subplots(figsize=(10,5))
sns.heatmap(data.corr(method='pearson'), ax=ax, annot=True, fmt='.3f')
```

```
<AxesSubplot:>
```



Предсказание целевого признака

Предскажем значение целевого признака L.

Разделение выборки на обучающую и тестовую

```
X = data.drop(columns='L')
Y = data['L']
```

Входные данные:

```
X.head()
```

	Temperature	R	A_M	Type	Color_cat	Spectral_Class_cat
0	3068	0.1700	16.12	0	8	5
1	3042	0.1542	16.60	0	8	5
2	2600	0.1020	18.70	0	8	5
3	2800	0.1600	16.65	0	8	5
4	1939	0.1030	20.06	0	8	5

Выходные данные:

```
Y.head()
```

```
0    0.002400
1    0.000500
2    0.000300
```

```
3    0.000200
4    0.000138
Name: L, dtype: float64
```

```
X_train,  X_test,  Y_train,  Y_test = train_test_split(X,  Y,
random_state = 2023, test_size = 0.1)
```

Входные параметры обучающей выборки

```
X_train.head()
```

	Temperature	R	A_M	Type	Color_cat	Spectral_Class_cat
192	2994	0.28000	13.45	1	8	5
123	3146	0.09320	16.92	0	8	5
140	13420	0.00981	13.67	2	1	1
8	2650	0.11000	17.45	0	8	5
30	39000	10.60000	-4.70	3	0	6

Выходные параметры обучающей выборки

```
Y_train.head()
```

```
192         0.00720
123         0.00015
140         0.00059
8           0.00069
30     204000.00000
Name: L, dtype: float64
```

Выходные параметры тестовой выборки

```
Y_test.head()
```

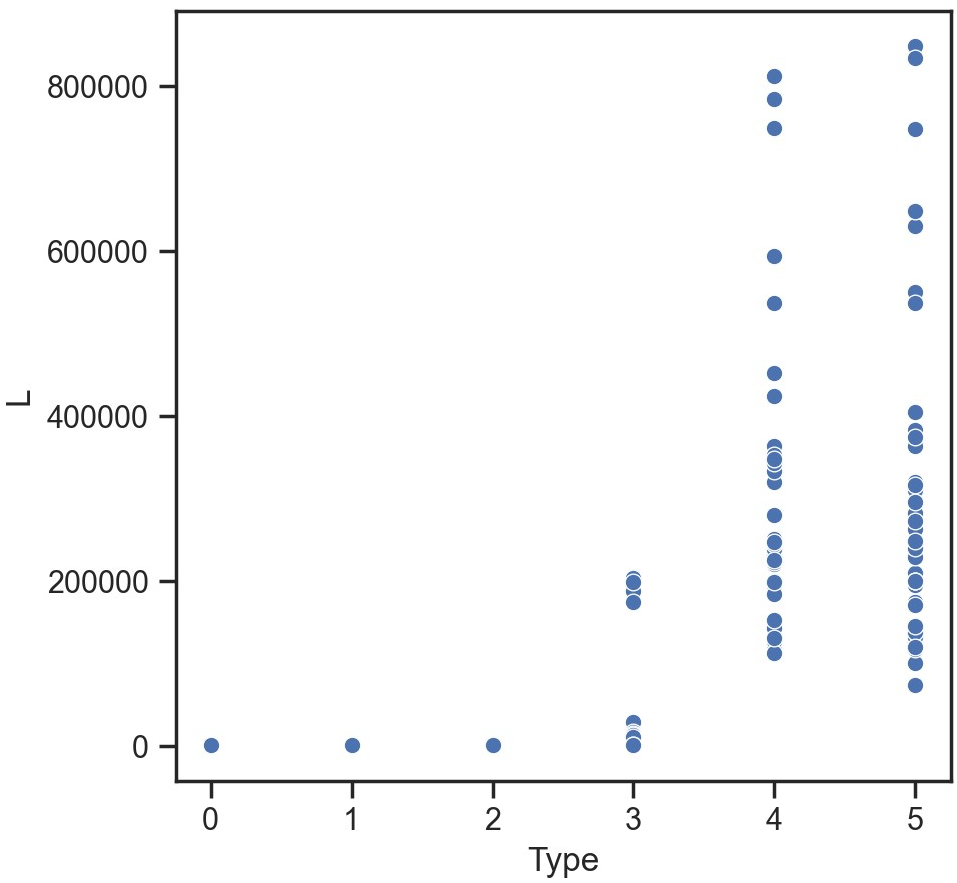
```
42     150000.000000
205         0.001560
4           0.000138
120         0.000430
74          0.004000
Name: L, dtype: float64
```

SVM

```
from sklearn.svm import SVR , LinearSVR
from sklearn.datasets import make_blobs
from sklearn.metrics import mean_absolute_error, mean_squared_error
```

```
fig, ax = plt.subplots(figsize=(5,5))
sns.scatterplot(ax=ax, x=X['Type'], y=Y)
```

```
<AxesSubplot:xlabel='Type', ylabel='L'>
```




```
svr_1 = SVR()
svr_1.fit(X_train, Y_train)
```

```
SVR()
```

```
Y_pred_1 = svr_1.predict(X_test)
```

Проверим результат на 2 метриках

```
mean_absolute_error(Y_test, Y_pred_1), mean_squared_error(Y_test,
Y_pred_1)
```

```
(51078.38318243809, 12038640007.631216)
```

Случайный лес

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.tree import DecisionTreeClassifier,
DecisionTreeRegressor, export_graphviz
```

Обучим регрессор на 4 деревьях

```
tree1 = RandomForestRegressor(n_estimators=4, oob_score=True,
random_state=2023)
tree1.fit(X, Y)
```

```
C:\Users\79626\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\ensemble\_forest.py:583: UserWarning: Some inputs do
not have OOB scores. This probably means too few trees were used to
compute any reliable OOB estimates.
  warn(
```

```
RandomForestRegressor(n_estimators=4, oob_score=True,
random_state=2023)
```

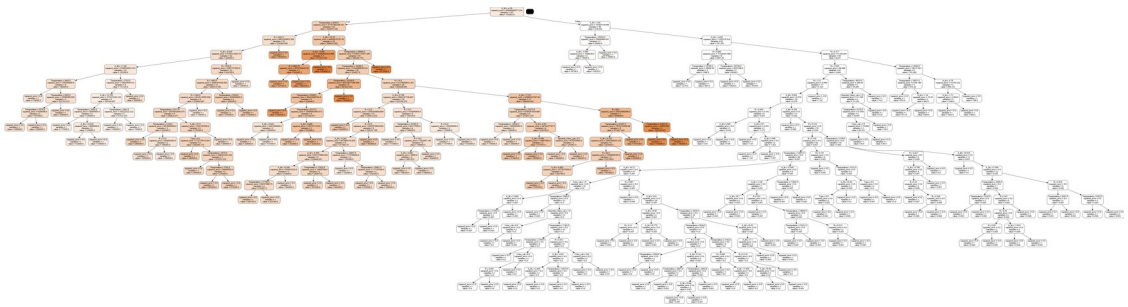
```
# Out-of-bag error, возвращаемый регрессором
tree1.oob_score_, 1-tree1.oob_score_
```

```
(0.40162426738193213, 0.5983757326180679)
```

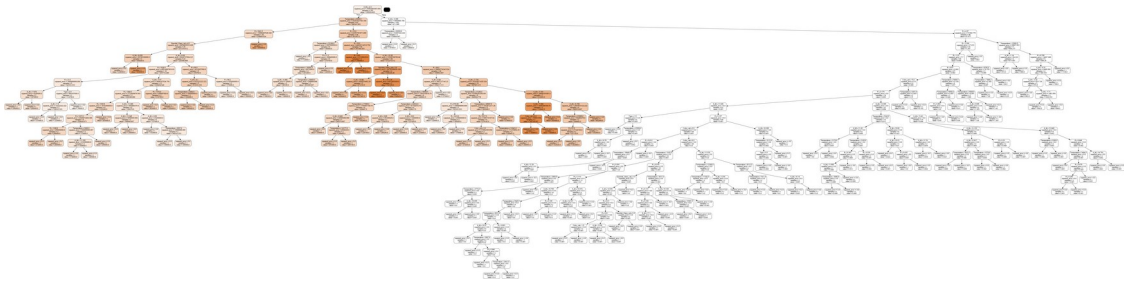
Визуализируем обучающие деревья

```
# Визуализация дерева
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()
```

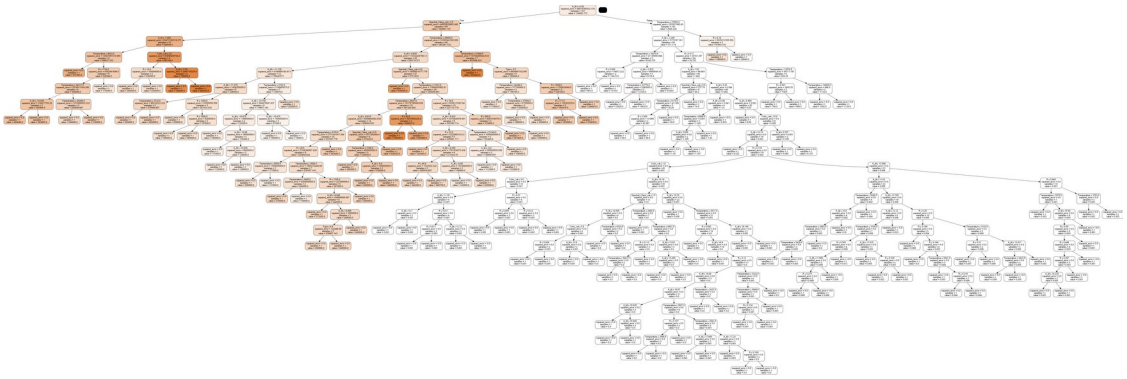
```
Image(get_png_tree(tree1.estimators_[0], X.columns), width="500")
```



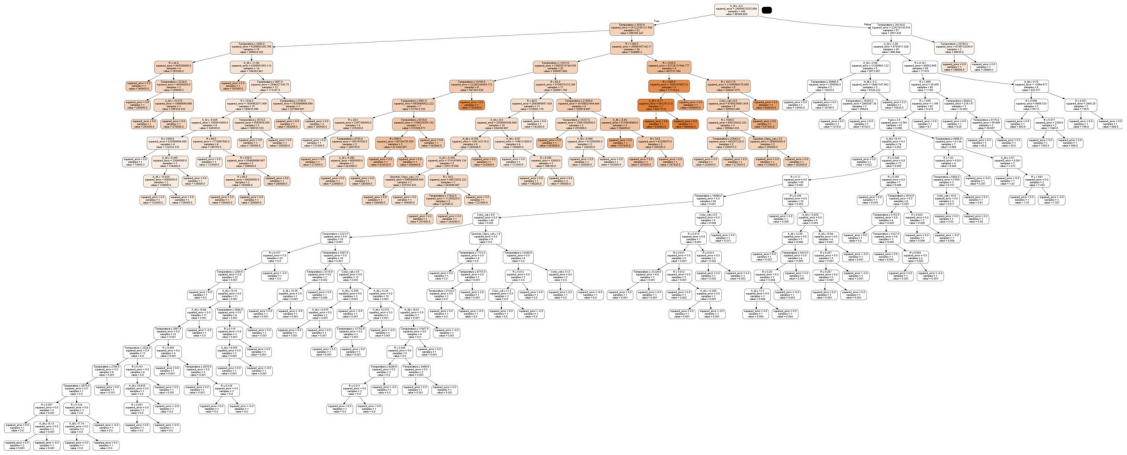
```
Image(get_png_tree(tree1.estimators_[1], X.columns), width="500")
```



```
Image(get_png_tree(tree1.estimators_[2], X.columns), width="500")
```



```
Image(get_png_tree(tree1.estimators_[3], X.columns), width="500")
```



```
regressor = RandomForestRegressor(n_estimators=4, random_state=2022)
regressor.fit(X_train, Y_train)
y_pred = regressor.predict(X_test)

print('Mean Absolute Error:', mean_absolute_error(Y_test, y_pred))
print('Mean Squared Error:', mean_squared_error(Y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(mean_squared_error(Y_test,
y_pred)))
```

```
Mean Absolute Error: 36182.99189315625
Mean Squared Error: 7104567321.66899
Root Mean Squared Error: 84288.59544249738
```

Вывод:

Как видно, случайный лес показало намного более лучшие результаты, чем линейная регрессия. Основная причина в отсутствии масштабировавния данных (в обоих случаях).