Московский государственный технический

университет им. Н. Э. Баумана

Отчёт по лабораторной работе №3 по курсу «Технологии машинного обучения».

«Подготовка обучающей и тестовой выборки, кросс-валидация и подбор гиперпараметров на примере метода ближайших соседей».

Выполнил:
Головацкий А. Д.
студент группы ИУ5-61Б

Проверил:
Гапанюк Ю. Е.

Подпись и дата:

Подпись и дата:

Москва, 2022 г.

ИУ5-61Б Головацкий Андрей

Подготовка обучающей и тестовой выборки, кросс-валидация и подбор гиперпараметров на примере метода ближайших соседей

1. Выберите набор данных (датасет) для решения задачи классификации или регрессии.
2. С использованием метода train_test_split разделите выборку на обучающую и тестовую.
3. Обучите модель ближайших соседей для произвольно заданного гиперпараметра K. Оцените качество модели с помощью подходящих для задачи метрик.
4. Произведите подбор гиперпараметра K с использованием GridSearchCV и/или RandomizedSearchCV и кросс-валидации, оцените качество оптимальной модели. Желательно использование нескольких стратегий кросс-валидации.
5. Сравните метрики качества исходной и оптимальной моделей.

```
import pandas as pd
import sklearn
from sklearn.model_selection import train_test_split
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error, r2_score
import seaborn as sns
from sklearn.neighbors import KNeighborsRegressor
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import KFold, RepeatedKFold
```

Датасет будет представлять собой набор лучших городов для работы-жизни в 2022 году.

```
df = pd.read_csv("C:\\Users\\Andrew\\Anaconda Projects\\datasets\\
cities-work-life-balance-2022.csv")

#Удаление строк, содержащих пустые значения
df = df.dropna()
df.head()
```

```
   2022 2021        City      Country Remote Jobs Overworked
Population  \
0     1    2        Oslo       Norway      41.72%
11.20%
1     2    -        Bern  Switzerland      44.86%
11.40%
2     3    1    Helsinki      Finland      38.92%
12.70%
```

```
3     4    3      Zurich  Switzerland      44.86%
11.90%
4     5    5  Copenhagen      Denmark      41.42%
10.50%

   Minimum Vacations Offered (Days) Vacations Taken (Days)
Unemployment  \
0                                 25                     25
94.7
1                                 20                     25
99.8
2                                 25                     30
89.3
3                                 20                     25
99.2
4                                 25                     28
94.8

  Multiple Jobholders  ... Healthcare Access to Mental Healthcare  \
0               9.10%  ...      100.0                         85.0
1               7.60%  ...       99.6                         78.6
2               6.30%  ...       96.7                         73.0
3               7.60%  ...       99.2                         78.6
4               7.60%  ...       94.8                         77.6

   Inclusivity & Tolerance  Affordability  Happiness, Culture &
Leisure  \
0                    93.2           59.4
88.8
1                    94.6           69.9
100.0
2                    93.9           65.0
96.3
3                    87.5           71.6
91.5
4                    95.2           65.3
92.5

   City Safety  Outdoor Spaces  Air Quality  Wellness and Fitness
TOTAL SCORE
0         86.5            95.6         97.5                  65.7
100.00
1         91.8            87.1        100.0                  69.1
99.46
2         94.9            86.0         97.0                  68.3
99.24
3         92.8            84.0         96.2                  68.7
96.33
4         95.7            75.5         95.1                  66.3
96.21
```

```
[5 rows x 24 columns]
```

```
df.shape
```

```
(100, 24)
```

Обучающая и тестовая выборка

- В качестве обучающей выборки будем использовать процент работ, которые можно выполнять удаленно в каждом городе
- В качестве целевого признака будет выступать общий счет города
- Алгоритм KNN будет решать задачу регрессии

```
df["Remote Jobs"]
```

```
0     41.72%
1     44.86%
2     38.92%
3     44.86%
4     41.42%
       ...
95    16.84%
96    25.65%
97    30.70%
98    28.89%
99    26.06%
Name: Remote Jobs, Length: 100, dtype: object
```

```
#Преобразование в удобный формат
rj = []
for item in df["Remote Jobs"]:
    rj.append(float(item.strip('%')))
myData = pd.DataFrame({"rj": rj, "h": df["Access to Mental
Healthcare"], "score": df["TOTAL SCORE"]})
myData
```

```
       rj     h   score
0   41.72  85.0  100.00
1   44.86  78.6   99.46
2   38.92  73.0   99.24
3   44.86  78.6   96.33
4   41.42  77.6   96.21
..    ...   ...     ...
95  16.84  79.7   70.73
96  25.65  50.0   66.57
97  30.70  74.3   66.02
98  28.89  52.2   61.23
99  26.06  65.0   50.00

[100 rows x 3 columns]
```

```
df['TOTAL SCORE'].corr(pd.Series(rj))
```

```
0.5859739478939351
```

```
df['TOTAL SCORE'].corr(df['Access to Mental Healthcare'])
```

```
0.5292511308951154
```

```
#С помощью метода train_test_split разделим выборку на обучающую и
тестовую
data=np.array([[myData["h"].iloc[i], myData["rj"].iloc[i]] for i in
range(myData.shape[0])])
target= np.array(myData["score"])
trainX, testX, trainY, testY = train_test_split(data, target,
train_size=0.2, random_state=1)
data
```

```
array([[ 85.  ,  41.72],
       [ 78.6 ,  44.86],
       [ 73.  ,  38.92],
       [ 78.6 ,  44.86],
       [ 77.6 ,  41.42],
       [ 78.6 ,  44.86],
       [ 92.4 ,  37.81],
       [ 67.4 ,  38.79],
       [ 82.  ,  36.73],
       [ 82.  ,  36.73],
       [ 77.1 ,  44.2 ],
       [ 67.4 ,  38.79],
       [ 85.1 ,  41.55],
       [ 84.  ,  36.52],
       [ 82.  ,  36.73],
       [ 92.4 ,  37.81],
       [ 76.6 ,  36.25],
       [ 82.  ,  36.73],
       [ 92.4 ,  37.81],
       [ 82.  ,  36.73],
       [ 82.  ,  36.73],
       [ 82.  ,  36.73],
       [ 82.  ,  36.73],
       [ 83.8 ,  43.5 ],
       [ 83.8 ,  43.5 ],
       [ 82.  ,  36.73],
       [ 83.8 ,  43.5 ],
       [ 79.3 ,  37.74],
       [ 77.3 ,  36.69],
       [ 92.4 ,  37.81],
       [ 83.8 ,  43.5 ],
       [ 65.7 ,  42.28],
       [ 66.8 ,  38.49],
       [ 77.3 ,  36.69],
```

```
[ 66.6 ,  41.14],
[ 66.  ,  43.34],
[ 70.6 ,  38.71],
[ 68.2 ,  44.35],
[ 70.9 ,  44.76],
[ 81.5 ,  42.34],
[ 67.8 ,  49.77],
[ 64.2 ,  31.69],
[ 66.4 ,  38.9 ],
[ 66.6 ,  39.52],
[ 62.1 ,  52.06],
[ 66.7 ,  42.58],
[ 67.1 ,  34.64],
[ 67.5 ,  41.3 ],
[ 66.6 ,  40.34],
[ 64.2 ,  31.69],
[ 76.6 ,  33.16],
[ 66.3 ,  42.66],
[ 68.1 ,  35.43],
[ 68.1 ,  30.92],
[ 65.7 ,  40.31],
[ 67.2 ,  39.94],
[ 67.  ,  39.48],
[ 66.6 ,  37.87],
[ 68.1 ,  41.96],
[ 66.9 ,  40.39],
[ 66.9 ,  36.48],
[ 67.5 ,  45.51],
[ 73.1 ,  34.99],
[ 66.8 ,  36.82],
[ 66.4 ,  36.61],
[ 67.2 ,  36.41],
[ 66.6 ,  36.2 ],
[ 66.4 ,  39.54],
[ 68.  ,  39.18],
[ 67.2 ,  36.7 ],
[ 69.1 ,  41.55],
[ 66.8 ,  40.11],
[ 66.8 ,  35.96],
[ 67.7 ,  37.29],
[ 66.6 ,  31.31],
[ 65.6 ,  37.34],
[ 67.5 ,  39.69],
[ 66.5 ,  39.09],
[ 67.  ,  38.88],
[ 67.5 ,  37.71],
[ 65.6 ,  35.13],
[ 66.1 ,  33.54],
[ 67.  ,  33.01],
[ 66.5 ,  35.07],
```

```
        [ 66.6 ,  35.72],
        [ 67.5 ,  34.83],
        [ 67.5 ,  37.33],
        [ 75.1 ,  37.  ],
        [ 65.7 ,  30.41],
        [ 67.8 ,  41.57],
        [ 66.7 ,  37.59],
        [ 67.2 ,  31.81],
        [100.  ,  40.27],
        [ 64.6 ,  27.29],
        [ 70.2 ,  28.39],
        [ 79.7 ,  16.84],
        [ 50.  ,  25.65],
        [ 74.3 ,  30.7 ],
        [ 52.2 ,  28.89],
        [ 65.  ,  26.06]])
```

```python
#Обучение модели с гиперпараметром 3
neigh = KNeighborsRegressor(n_neighbors=3)
neigh.fit(trainX, trainY)
pred3_1 = neigh.predict(testX)
pred3_2 = neigh.predict(trainX)
pred3_1 = [i for  i in pred3_1]
pred3_2 = [i for  i in pred3_2]

#Обучение модели с гиперпараметром 10
neigh = KNeighborsRegressor(n_neighbors=10)
neigh.fit(trainX, trainY)
pred10_1 = neigh.predict(testX)
pred10_2 = neigh.predict(trainX)
pred10_1 = [i for  i in pred10_1]
pred10_2 = [i for  i in pred10_2]

#Обучение модели с гиперпараметром 40
neigh = KNeighborsRegressor(n_neighbors=70)
neigh.fit(trainX, trainY)
pred40_1 = neigh.predict(testX)
pred40_2 = neigh.predict(trainX)
pred40_1 = [i for  i in pred40_1]
pred40_2 = [i for  i in pred40_2]

print("Истинные значения:")
for i in testY:
    print(i)
```

```
Истинные значения:
81.4
79.91
88.17
80.87
74.59
```

91.79
87.64
80.78
82.83
83.25
77.11
87.18
84.27
84.62
84.77
88.2
88.38
85.66
81.46
93.31
99.24
81.85
66.02
83.66
91.06
87.83
73.15
89.73
85.3
87.34
83.05
50.0
84.56
70.73
79.47
87.1
84.94
84.06
90.68
88.09
79.89
84.56
81.62
92.23
80.36
86.85
85.63
77.12
89.74
61.23
85.77
84.31
90.68
96.21
84.23

```
84.93
90.83
79.71
96.33
81.81
89.1
83.15
82.77
86.01
85.29
79.11
93.79
83.89
100.0
77.15
84.24
90.73
83.67
83.63
94.04
66.57
92.47
82.92
79.9
92.45
```

```
print("Предсказания с гиперпараметром 3:")
for i in pred3_1:
    print(i)
```

```
Предсказания с гиперпараметром 3:
82.38666666666667
82.38666666666667
88.79
82.38666666666667
82.38666666666667
91.55666666666666
85.32
82.38666666666667
82.29
82.29
92.00333333333333
95.96666666666665
85.60333333333334
82.29
83.87333333333333
85.32
83.87333333333333
83.87333333333333
85.60333333333334
94.89333333333332
```

87.74333333333334
85.85000000000001
88.79
88.79
91.55666666666666
83.87333333333333
83.42999999999999
90.14
82.38666666666667
83.87333333333333
85.60333333333334
82.38666666666667
85.60333333333334
88.79
82.38666666666667
83.87333333333333
85.60333333333334
85.60333333333334
95.96666666666665
85.60333333333334
85.85000000000001
82.29
85.60333333333334
92.00333333333333
82.38666666666667
82.38666666666667
83.87333333333333
82.38666666666667
95.96666666666665
82.38666666666666
85.60333333333334
85.60333333333334
95.96666666666665
94.89333333333332
83.87333333333333
82.38666666666667
91.55666666666666
88.79
95.96666666666665
82.38666666666667
95.96666666666665
82.38666666666667
83.87333333333333
85.60333333333334
85.60333333333334
85.60333333333334
91.55666666666666
82.29
91.21333333333332
82.17999999999999

```
85.32333333333332
91.55666666666666
83.87333333333333
82.29
85.32
82.38666666666666
91.55666666666666
85.60333333333334
82.29
91.55666666666666
```

```
print("Предсказания с гиперпараметром 40:")
for i in pred40_1:
    print(i)
```

```
Предсказания с гиперпараметром 40:
53.55714285714286
52.34285714285714
47.042857142857144
53.84285714285714
56.31428571428572
44.44285714285714
51.55714285714286
53.84285714285714
52.41428571428571
52.357142857142854
```

Проверим качество модели с помощью метрик регрессии

Метрика Root mean squared error (RMSE)

```
myrmse3_1 = mean_squared_error(testY, pred3_1)
myrmse3_2 = mean_squared_error(trainY, pred3_2)
print("Для K=3:\t {}\t{}".format(myrmse3_2, myrmse3_1))
```

```
Для K=3:    9.153681666666696    49.71477541666667
```

```
myrmse10_1 = mean_squared_error(testY, pred10_1)
myrmse10_2 = mean_squared_error(trainY, pred10_2)
print("Для K=10:\t {}\t{}".format(myrmse10_2, myrmse10_1))
```

```
Для K=10:   12.008734900000018   55.63536254999998
```

```
#Запоминаем результаты
rmse = []
rmse.append(myrmse10_2)
rmse.append(myrmse10_1)
```

Коэффициент детерминации

```
r2_1 = r2_score(testY,pred3_1)
r2_2 = r2_score(trainY,pred3_2)
print("Для K=3:\t {}\t{}".format(r2_1, r2_2))
```

```
Для K=3:    0.17609743659179944  0.6800162595892234
```

```
r2_1 = r2_score(testY,pred10_1)
r2_2 = r2_score(trainY,pred10_2)
print("Для K=10:\t {}\t{}".format(r2_1, r2_2))
```

```
Для K=10:   0.07797797662940764  0.5802126345625127
```

Подбор гиперпараметра с помощью кросс-валидации

Найдём наилучший гиперпараметр используя текущую перестановку

```
sorted(sklearn.metrics.SCORERS.keys())
```

```
['accuracy',
 'adjusted_mutual_info_score',
 'adjusted_rand_score',
 'average_precision',
 'balanced_accuracy',
 'brier_score_loss',
 'completeness_score',
 'explained_variance',
 'f1',
 'f1_macro',
 'f1_micro',
 'f1_samples',
 'f1_weighted',
 'fowlkes_mallows_score',
 'homogeneity_score',
 'jaccard',
 'jaccard_macro',
 'jaccard_micro',
 'jaccard_samples',
 'jaccard_weighted',
 'max_error',
 'mutual_info_score',
 'neg_log_loss',
 'neg_mean_absolute_error',
 'neg_mean_squared_error',
 'neg_mean_squared_log_error',
 'neg_median_absolute_error',
 'normalized_mutual_info_score',
 'precision',
 'precision_macro',
 'precision_micro',
 'precision_samples',
 'precision_weighted',
 'r2',
 'recall',
 'recall_macro',
 'recall_micro',
```

```
 'recall_samples',
 'recall_weighted',
 'roc_auc',
 'v_measure_score']
```

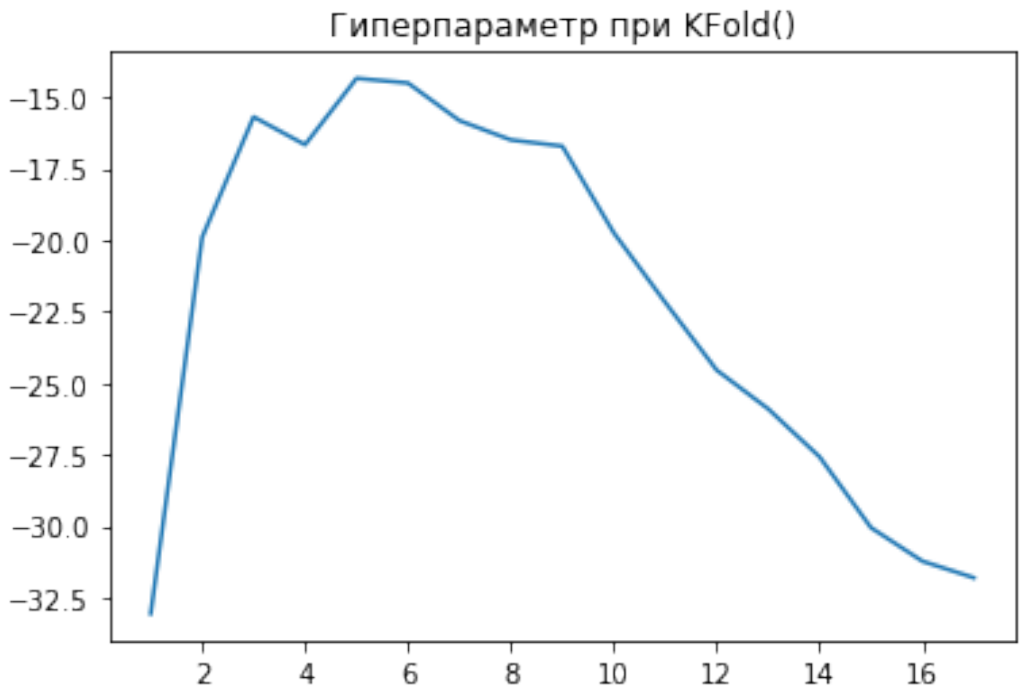
KFold()

```
kf = KFold(n_splits=10)
r = np.array(range(1,18))
params = [{"n_neighbors": r}]
gs= GridSearchCV(KNeighborsRegressor(), params, cv=kf,
scoring="neg_mean_squared_error")
gs.fit(trainX, trainY)
```

```
GridSearchCV(cv=KFold(n_splits=10, random_state=None, shuffle=False),
             error_score='raise-deprecating',
             estimator=KNeighborsRegressor(algorithm='auto',
leaf_size=30,
                                           metric='minkowski',
                                           metric_params=None,
n_jobs=None,
                                           n_neighbors=5, p=2,
                                           weights='uniform'),
             iid='warn', n_jobs=None,
             param_grid=[{'n_neighbors': array([ 1,  2,  3,  4,  5,
6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17])}],
             pre_dispatch='2*n_jobs', refit=True,
return_train_score=False,
             scoring='neg_mean_squared_error', verbose=0)
```

```
plt.plot(r, gs.cv_results_['mean_test_score'])
plt.title("Гиперпараметр при KFold()")
```

```
Text(0.5, 1.0, 'Гиперпараметр при KFold()')
```




```
#Предсказание результатов на тестовой и обучающей выборках
gs.best_estimator_.fit(trainX, trainY)
predGs1 = gs.best_estimator_.predict(testX)
predGs2 = gs.best_estimator_.predict(trainX)

#Оценка качества с помощью RMSE
rmse.append(mean_squared_error(testY, predGs1))
rmse.append(mean_squared_error(trainY, predGs2))
(mean_squared_error(testY, predGs1), mean_squared_error(trainY,
predGs2))
```

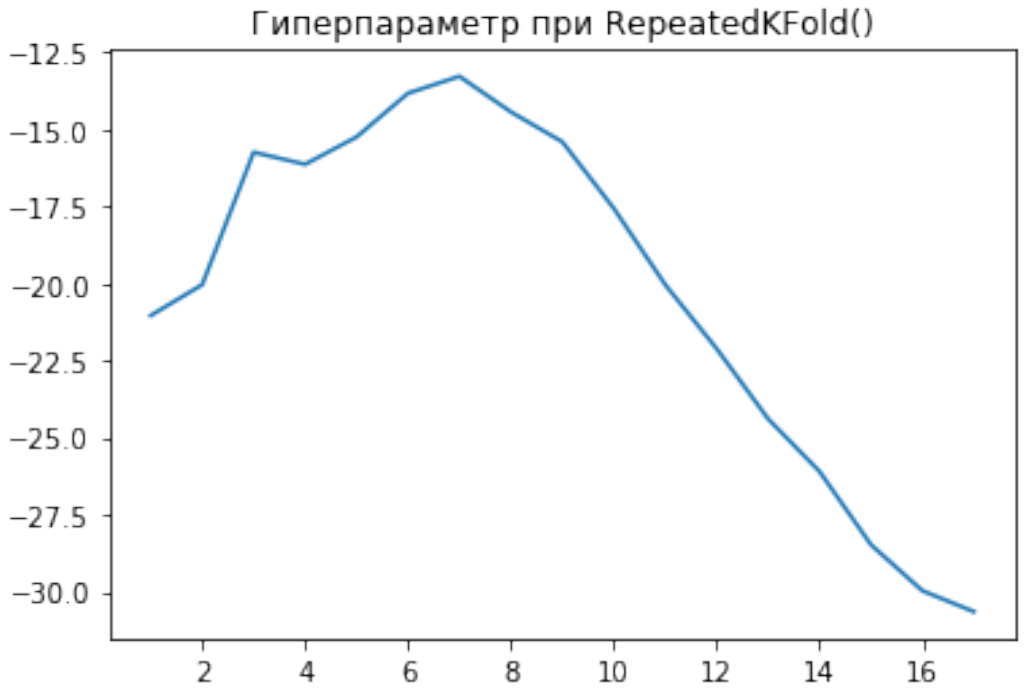
```
(51.03391104999997, 10.210094800000014)
```

RepeatedKFold()

```
kf = RepeatedKFold(n_splits=10, n_repeats = 3)
r = np.array(range(1,18))
params = [{"n_neighbors": r}]
gs2= GridSearchCV(KNeighborsRegressor(), params, cv=kf,
scoring="neg_mean_squared_error")
gs2.fit(trainX, trainY)
```

```
GridSearchCV(cv=<sklearn.model_selection._split.RepeatedKFold object
at 0x000001B0CBFBD488>,
             error_score='raise-deprecating',
             estimator=KNeighborsRegressor(algorithm='auto',
leaf_size=30,
                                           metric='minkowski',
                                           metric_params=None,
n_jobs=None,
```

```
                                             n_neighbors=5, p=2,
                                             weights='uniform'),
             iid='warn', n_jobs=None,
             param_grid=[{'n_neighbors': array([ 1,  2,  3,  4,  5,
6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17])}],
             pre_dispatch='2*n_jobs', refit=True,
return_train_score=False,
             scoring='neg_mean_squared_error', verbose=0)

plt.plot(r, gs2.cv_results_['mean_test_score'])
plt.title("Гиперпараметр при RepeatedKFold()")

Text(0.5, 1.0, 'Гиперпараметр при RepeatedKFold()')
```



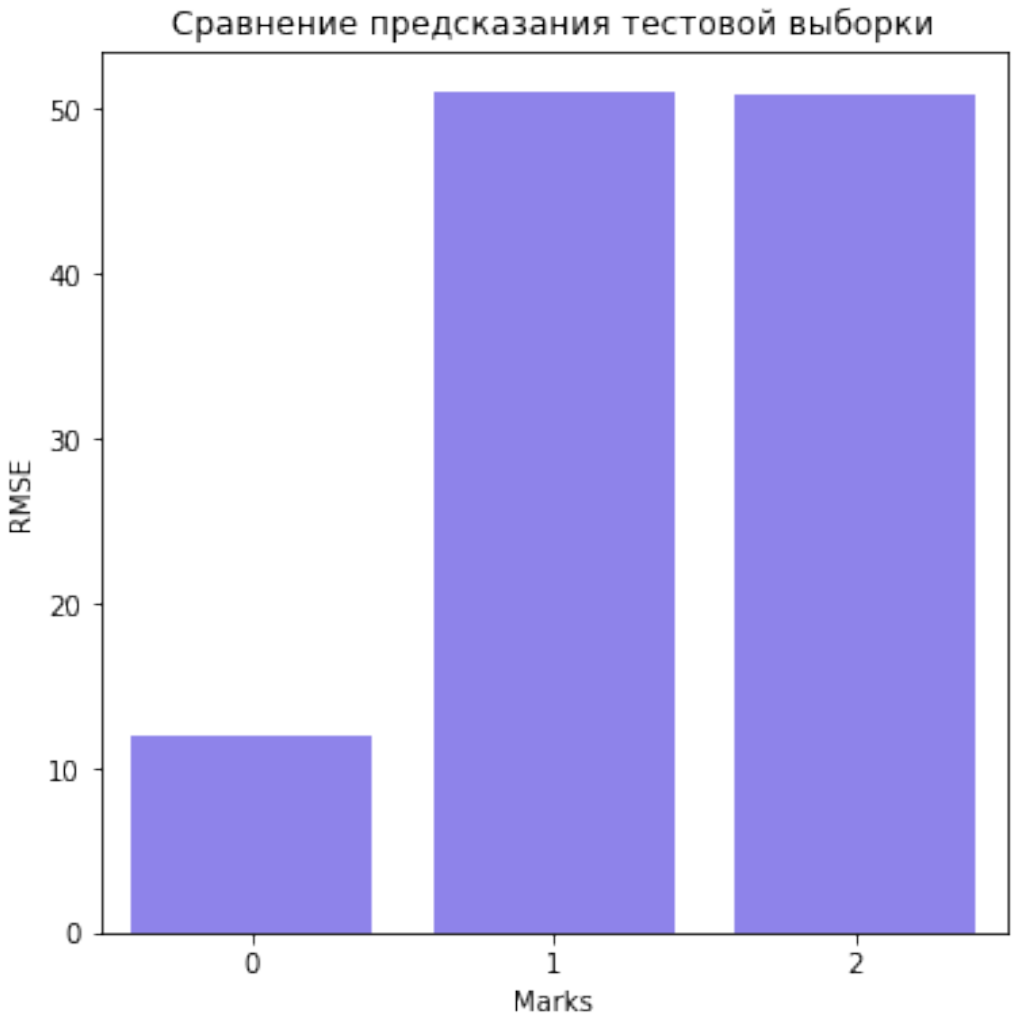

```
#Предсказание результатов на тестовой и обучающей выборках
gs2.best_estimator_.fit(trainX, trainY)
predGs1 = gs2.best_estimator_.predict(testX)
predGs2 = gs2.best_estimator_.predict(trainX)

#Оценка качества с помощью RMSE
rmse.append(mean_squared_error(testY, predGs1))
rmse.append(mean_squared_error(trainY, predGs2))
(mean_squared_error(testY, predGs1), mean_squared_error(trainY,
predGs2))

(50.82131951530611, 10.416889285714312)
```

Сравним метрики качества исходной и оптимальных моделей

```
X = [i for i in range(len(rmse)//2)]
testRMSE = []
```

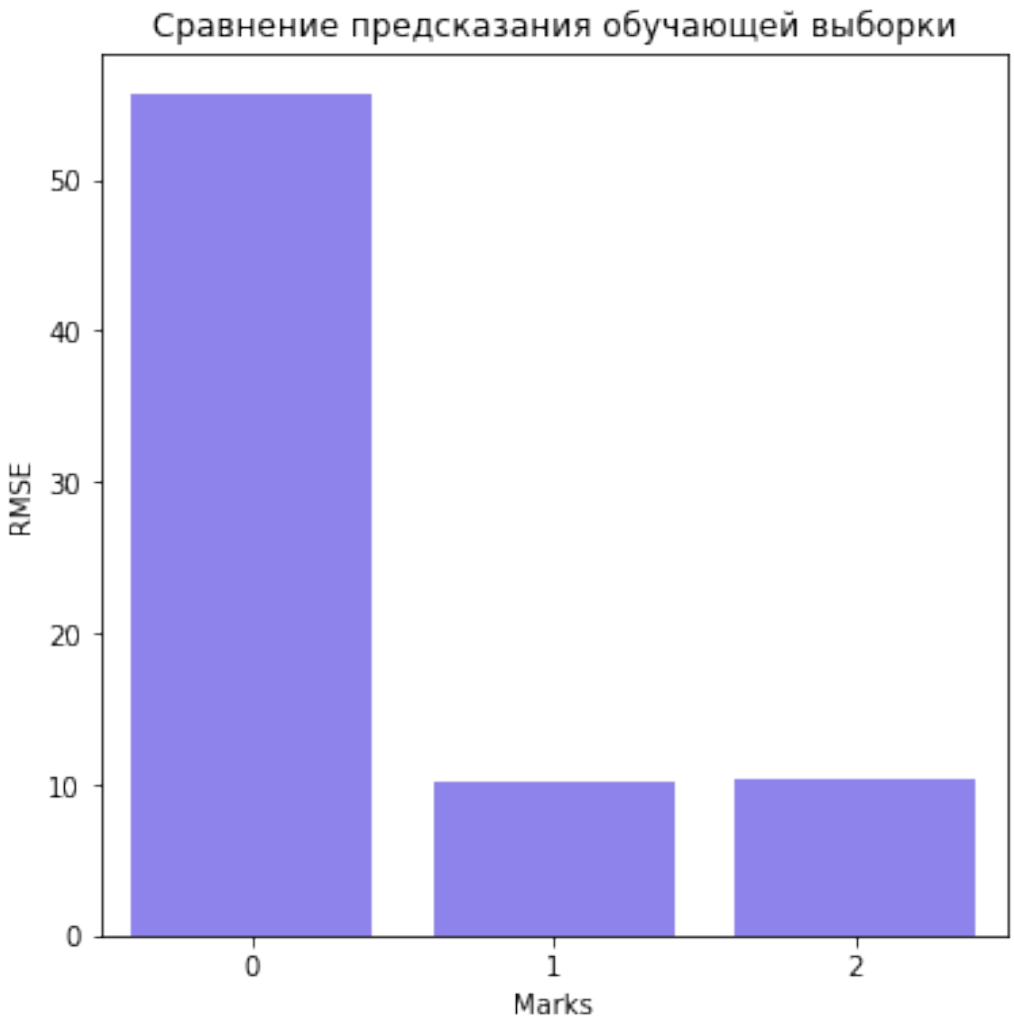
```
trainRMSE = []
for i in range(len(rmse)):
   if i%2==0:
        trainRMSE.append(rmse[i])
   else:
        testRMSE.append(rmse[i])

dataframe = pd.DataFrame({"Marks": X, "RMSE": trainRMSE})
fig, ax = plt.subplots(figsize=(6,6))
ax.title.set_text("Сравнение предсказания тестовой выборки")
sns.barplot(data=dataframe, y="RMSE", x="Marks", color="#8172fb")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1b0cbf87688>
```




```
dataframe = pd.DataFrame({"Marks": X, "RMSE": testRMSE})
fig, ax = plt.subplots(figsize=(6,6))
ax.title.set_text("Сравнение предсказания обучающей выборки")
sns.barplot(data=dataframe, y="RMSE", x="Marks", color="#8172fb")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1b0cbfdb8c8>
```


Сравнение предсказания обучающей выборки
RMSE
0
10
20
30
40
50
0
1
2
Marks