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)

ФАКУЛЬТЕТ Информатика и системы управления
КАФЕДРА Системы обработки информации и управления

Отчет по лабораторной работе №4

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

по курсу «Технологии машинного обучения»

Выполнил:
Студент группы ИУ5Ц-81Б
Тураев Глеб

Проверил:
Преподаватель кафедры ИУ5
Гапанюк Ю.Е.

Москва 2020

Цель лабораторной работы: Изучение способов предварительной обработки данных для дальнейшего формирования моделей.

Задание:

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. Постройте модель и оцените качество модели с использованием кросс-валидации.
5. Произведите подбор гиперпараметра K с использованием GridSearchCV и кросс-валидации.

Выполнение лабораторной работы:

Импортируем библиотеки:

Осуществим импорт библиотек с помощью команды **import:**

```
[63] import numpy as np
     import pandas as pd

     from typing import Dict, Tuple
     from scipy import stats
     from sklearn.datasets import load_wine
     from sklearn.model_selection import train_test_split, cross_val_score, cross_validate, GridSearchCV
     from sklearn.neighbors import KNeighborsRegressor, KNeighborsClassifier
     from sklearn.metrics import *

     import seaborn as sns
     import matplotlib.pyplot as plt
     %matplotlib inline
     sns.set(style="ticks")
```

```
[64] wine = load_wine()
```

```
[65] #наименование признаков
     wine.feature_names
```

```
['alcohol',
 'malic_acid',
 'ash',
 'alcalinity_of_ash',
 'magnesium',
 'total_phenols',
 'flavanoids',
 'nonflavanoid_phenols',
 'proanthocyanins',
 'color_intensity',
 'hue',
 'od280/od315_of_diluted_wines',
 'proline']
```

```
[66] #узнаем размер датасета
     wine.data.shape
```

```
(178, 13)
```

Формирование DataFrame:

```
[67] wine_df = pd.DataFrame(data=np.c_[wine['data'], wine['target']], columns = list(wine['feature_names']) + ['target'])
```

```
[68] wine_df.describe()
```

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavanoids	nonflavanoid_phenols	proanthocyanins	color_intensity	hue	od280/od315_of_diluted_wines	proline	target
count	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000
mean	13.000618	2.336348	2.366517	19.494944	99.741573	2.295112	2.029270	0.361854	1.590899	5.058090	0.957449	2.611685	746.893258	0.938202
std	0.811827	1.117146	0.274344	3.339564	14.282484	0.625851	0.998859	0.124453	0.572359	2.318286	0.228572	0.709990	314.907474	0.775035
min	11.030000	0.740000	1.360000	10.600000	70.000000	0.980000	0.340000	0.130000	0.410000	1.280000	0.480000	1.270000	278.000000	0.000000
25%	12.362500	1.602500	2.210000	17.200000	88.000000	1.742500	1.205000	0.270000	1.250000	3.220000	0.782500	1.937500	500.500000	0.000000
50%	13.050000	1.865000	2.360000	19.500000	98.000000	2.355000	2.135000	0.340000	1.555000	4.690000	0.965000	2.780000	673.500000	1.000000
75%	13.677500	3.082500	2.557500	21.500000	107.000000	2.800000	2.875000	0.437500	1.950000	6.200000	1.120000	3.170000	985.000000	2.000000
max	14.830000	5.800000	3.230000	30.000000	162.000000	3.880000	5.080000	0.660000	3.580000	13.000000	1.710000	4.000000	1680.000000	2.000000

Разделение на обучающую и тестовую выборку:

```
[69] wine_X_train, wine_X_test, wine_Y_train, wine_Y_test = train_test_split(wine.data, wine.target, test_size = 0.3, random_state=1)
```

Размер обучающей выборки:

```
[70] wine_X_train.shape, wine_Y_train.shape
```

```
((124, 13), (124,))
```

Размер тестовой выборки:

```
[71] wine_X_test.shape, wine_Y_test.shape
```

```
((54, 13), (54,))
```

3 ближайших соседа:

```
[72] cl3 = KNeighborsClassifier(n_neighbors=3)
     cl3.fit(wine_X_train, wine_Y_train)
     target3 = cl3.predict(wine_X_test)
     len(target3), target3
```

```
(54, array([0, 1, 2, 1, 0, 1, 2, 0, 2, 1, 0, 2, 1, 0, 2, 1, 1, 0, 1, 0, 0, 1,
       2, 0, 0, 2, 0, 0, 0, 1, 1, 1, 1, 0, 2, 1, 1, 2, 1, 0, 0, 1, 2, 0,
       0, 0, 0, 0, 0, 0, 1, 2, 2, 0]))
```

5 ближайших соседей:

```
[73] cl5 = KNeighborsClassifier(n_neighbors=5)
     cl5.fit(wine_X_train, wine_Y_train)
     target5 = cl5.predict(wine_X_test)
     len(target5), target5
```

```
(54, array([1, 1, 2, 2, 0, 1, 2, 0, 2, 1, 0, 2, 1, 0, 2, 1, 1, 0, 1, 0, 0, 1,
       2, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 2, 1, 1, 2, 1, 0, 0, 1, 2, 0,
       0, 0, 0, 0, 0, 0, 1, 2, 2, 0]))
```

Метрики качества классификации:

```
[74] accuracy_score(wine_Y_test, target3)
```

```
0.7407407407407407
```

```
[75] accuracy_score(wine_Y_test, target5)
```

```
0.7037037037037037
```

Конвертация целевого признака в бинарный:

```
[76] def convert_target_to_binary(array:np.ndarray, target:int) -> np.ndarray:
         res = [1 if x == target else 0 for x in array]
         return res
     bin_wine_Y_test = convert_target_to_binary(wine_Y_test, 2)
     bin_target3=convert_target_to_binary(target3, 2)
     bin_target5=convert_target_to_binary(target5, 2)
     confusion_matrix(bin_wine_Y_test,bin_target3, labels=[0,1])
```

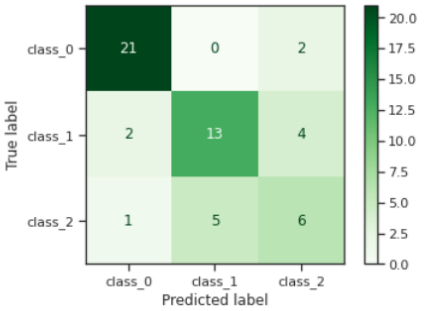
```
array([[36,  6],
       [ 6,  6]])
```

```
[77] tn, fp, fn, tp = confusion_matrix(bin_wine_Y_test, bin_target3).ravel()
     tn, fp, fn, tp
```

```
(36, 6, 6, 6)
```

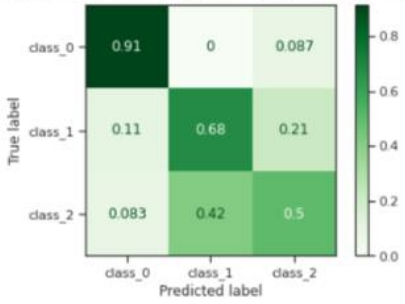
```
[78] plot_confusion_matrix(cl3, wine_X_test, wine_Y_test, display_labels = wine.target_names, cmap = plt.cm.Greens)
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7f9a49b9a240>
```



```
[79] plot_confusion_matrix(cl3, wine_X_test, wine_Y_test, display_labels = wine.target_names, cmap = plt.cm.Greens, normalize='true')
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7f9a4972ceb8>
```

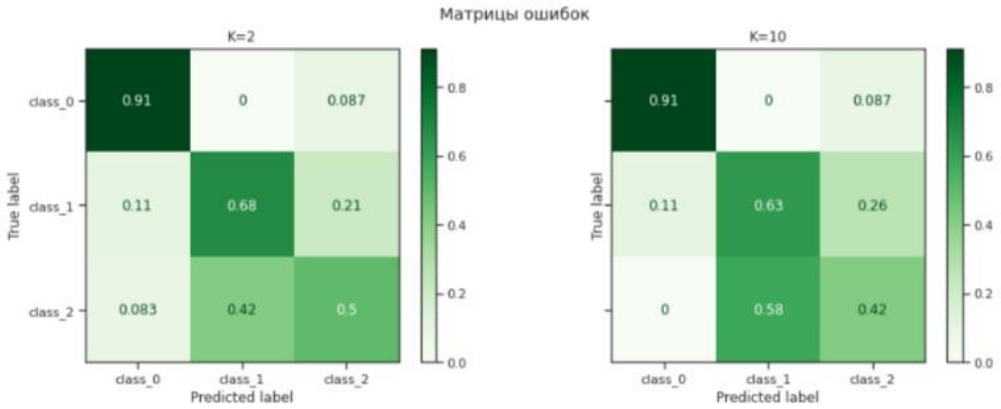



```
[80] fig, ax=plt.subplots(1, 2, sharex='col', sharey = 'row', figsize = (15,5))

     plot_confusion_matrix(cl3, wine_X_test, wine_Y_test, display_labels=wine.target_names, cmap=plt.cm.Greens, normalize='true', ax=ax[0])

     plot_confusion_matrix(cl5, wine_X_test, wine_Y_test, display_labels=wine.target_names, cmap=plt.cm.Greens, normalize='true', ax=ax[1])

     fig.suptitle("Матрицы ошибок")
     ax[0].title.set_text('K=2')
     ax[1].title.set_text('K=10')
```




```
[81] precision_score(bin_wine_Y_test, bin_target3), recall_score(bin_wine_Y_test, bin_target3)
```

```
(0.5, 0.5)
```

```
[82] precision_score(bin_wine_Y_test, bin_target5), recall_score(bin_wine_Y_test, bin_target5)
```

```
(0.4166666666666667, 0.4166666666666667)
```

С учетом веса класса:

```
[83] precision_score(wine_Y_test, target3, average = 'micro')
```

```
0.7407407407407407
```

Без учета веса класса:

```
[84] precision_score(wine_Y_test, target3, average = 'macro')
```

```
0.6990740740740741
```

С учетом веса классов:

```
[85] precision_score(wine_Y_test, target3, average = 'weighted')
```

```
0.7379115226337448
```

```
[86] classification_report(wine_Y_test, target3, target_names = wine.target_names,output_dict=True)
```

```
{'accuracy': 0.7407407407407407,
 'class_0': {'f1-score': 0.8936170212765957,
  'precision': 0.875,
  'recall': 0.9130434782608695,
  'support': 23},
 'class_1': {'f1-score': 0.7027027027027027,
  'precision': 0.7222222222222222,
  'recall': 0.6842105263157895,
  'support': 19},
 'class_2': {'f1-score': 0.5, 'precision': 0.5, 'recall': 0.5, 'support': 12},
 'macro avg': {'f1-score': 0.6987732413264328,
  'precision': 0.6990740740740741,
  'recall': 0.6990846681922197,
  'support': 54},
 'weighted avg': {'f1-score': 0.7389730155687603,
  'precision': 0.7379115226337448,
  'recall': 0.7407407407407407,
  'support': 54}}
```

ROC-кривая и ROC AUC:

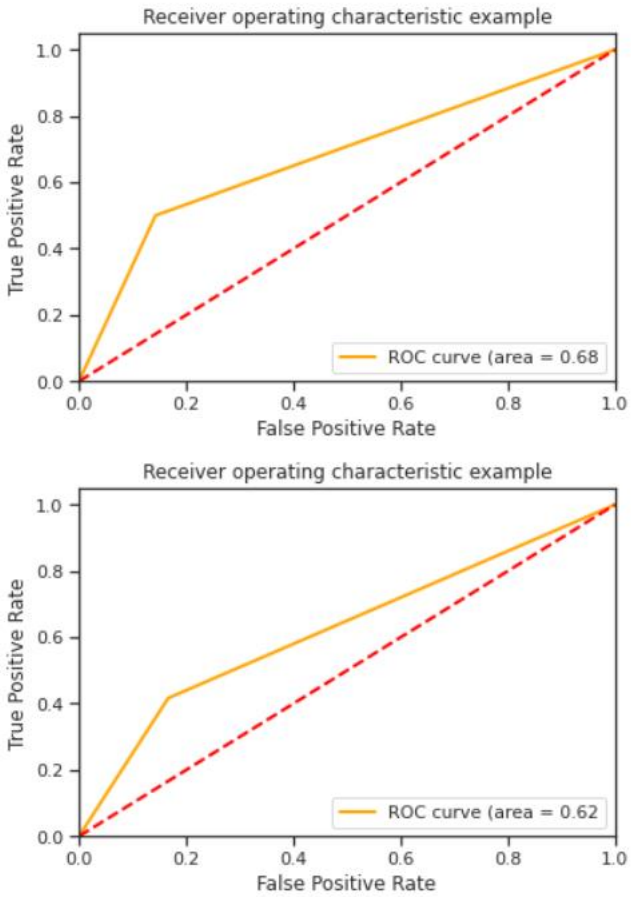
```
[87] fpr, tpr, thresholds = roc_curve(bin_wine_Y_test, bin_target3, pos_label=1)
     fpr, tpr, thresholds
```

```
(array([0.        , 0.14285714, 1.        ]),
 array([0. , 0.5, 1. ]),
 array([2, 1, 0]))
```

```
def draw_roc_curve(y_true, y_score, pos_label, average):
    fpr, tpr, thresholds = roc_curve(y_true, y_score, pos_label)
    roc_auc_value = roc_auc_score(y_true, y_score, average=average)
    plt.figure()
    lw=2
    plt.plot(fpr, tpr, color = 'orange', lw=lw, label = 'ROC curve (area = %0.2f' % roc_auc_value)
    plt.plot([0, 1], [0, 1], color = 'red', lw=lw, linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver operating characteristic example')
    plt.legend(loc='lower right')
    plt.show()

draw_roc_curve(bin_wine_Y_test, bin_target3, pos_label=1, average = 'micro')
draw_roc_curve(bin_wine_Y_test, bin_target5, pos_label=1, average = 'micro')
```

```
[88] draw_roc_curve(bin_wine_Y_test, bin_target3, pos_label=1, average = 'micro')
     draw_roc_curve(bin_wine_Y_test, bin_target5, pos_label=1, average = 'micro')
```



По полученным метрикам качества классификации можно говорить о среднем качестве классификации.

Разбиение выборки на K частей с помощью кросс-валидации. Стратегия кросс-валидации определяется автоматически (cross_val_score).

```
[89] wine_cross = cross_val_score(KNeighborsClassifier(n_neighbors=2), wine.data, wine.target, cv = 11)
     wine_cross
```

```
array([0.58823529, 0.64705882, 0.6875    , 0.5625    , 0.5625    ,
       0.625     , 0.8125    , 0.6875    , 0.8125    , 0.75      ,
       0.75      ])
```

```
[90] np.mean(wine_cross)
```

```
0.68048128342246
```

```
[91] wining = {
         'precesion': 'precision_weighted',
         'recall': 'recall_weighted',
         'f1': 'f1_weighted'
         }
     wine_cross = cross_validate(KNeighborsClassifier(n_neighbors=2), wine.data, wine.target, scoring=wining, cv=3, return_train_score=True)
     wine_cross
```

```
{'fit_time': array([0.00151014, 0.00049448, 0.00041747]),
 'score_time': array([0.00567651, 0.00400376, 0.00401187]),
 'test_f1': array([0.51069094, 0.6198816 , 0.6798559 ]),
 'test_precesion': array([0.48984127, 0.62317561, 0.70585516]),
 'test_recall': array([0.56666667, 0.6440678 , 0.72881356]),
 'train_f1': array([0.89415947, 0.8703245 , 0.8181316 ]),
 'train_precesion': array([0.91000807, 0.8877454 , 0.85825075]),
 'train_recall': array([0.89830508, 0.87394958, 0.83193277])}
```

Нахождение наилучшего параметра K с использованием GridSearchCV и кросс-валидации.

```
[92] n_range = np.array(range(5, 30 ,1))
     tuned_param = [{'n_neighbors': n_range}]
     tuned_param
```

```
[{'n_neighbors': array([ 5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21,
         22, 23, 24, 25, 26, 27, 28, 29])}]
```

```
[93] %%time
     classific_gs = GridSearchCV(KNeighborsClassifier(),tuned_param, cv=5, scoring='accuracy')
     classific_gs.fit(wine_X_train, wine_Y_train)
```

```
CPU times: user 292 ms, sys: 273 µs, total: 293 ms
Wall time: 295 ms
```

```
[94] classific_gs.cv_results_
```

```
{'mean_fit_time': array([0.00068803, 0.0004283 , 0.00043416, 0.00042524, 0.00041771,
       0.00042973, 0.0004241 , 0.000419  , 0.00047507, 0.00043502,
       0.00040317, 0.00039215, 0.00038939, 0.00040283, 0.00040059,
       0.00042143, 0.00041099, 0.000421  , 0.0004334 , 0.00040708,
       0.00040131, 0.00040536, 0.00039268, 0.00042624, 0.00040474]),
 'mean_score_time': array([0.00276728, 0.00158863, 0.00158939, 0.00154924, 0.00156426,
       0.00160794, 0.00169392, 0.00145273, 0.00176802, 0.00155969,
       0.0014349 , 0.00141783, 0.00144849, 0.00142608, 0.00149074,
       0.0015275 , 0.00144463, 0.0015048 , 0.00152011, 0.00152073,
       0.0014235 , 0.00145135, 0.00145826, 0.00145411, 0.00146427]),
 'mean_test_score': array([0.67666667, 0.725     , 0.66833333, 0.676     , 0.69266667,
       0.67666667, 0.67666667, 0.70033333, 0.70933333, 0.685     ,
       0.69333333, 0.71766667, 0.726     , 0.72633333, 0.726     ,
       0.734     , 0.734     , 0.75      , 0.726     , 0.718     ,
       0.71733333, 0.73366667, 0.709     , 0.71766667, 0.709     ]),
 'param_n_neighbors': masked_array(data=[5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19,
                    20, 21, 22, 23, 24, 25, 26, 27, 28, 29],
              mask=[False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False],
        fill_value='?',
             dtype=object),
 'params': [{'n_neighbors': 5},
  {'n_neighbors': 6},
  {'n_neighbors': 7},
  {'n_neighbors': 8},
  {'n_neighbors': 9},
  {'n_neighbors': 10},
  {'n_neighbors': 11},
```

```
[94] classific_gs.cv_results_
```

```
{'n_neighbors': 11},
{'n_neighbors': 12},
{'n_neighbors': 13},
{'n_neighbors': 14},
{'n_neighbors': 15},
{'n_neighbors': 16},
{'n_neighbors': 17},
{'n_neighbors': 18},
{'n_neighbors': 19},
{'n_neighbors': 20},
{'n_neighbors': 21},
{'n_neighbors': 22},
{'n_neighbors': 23},
{'n_neighbors': 24},
{'n_neighbors': 25},
{'n_neighbors': 26},
{'n_neighbors': 27},
{'n_neighbors': 28},
{'n_neighbors': 29}],
'rank_test_score': array([21,  9, 25, 24, 19, 22, 22, 17, 14, 20, 18, 11,  6,  5,  6,  2,  2,
        1,  6, 10, 13,  4, 15, 11, 15], dtype=int32),
'split0_test_score': array([0.64, 0.64, 0.6 , 0.6 , 0.64, 0.64, 0.64, 0.64, 0.64, 0.6 , 0.6 ,
       0.76, 0.76, 0.72, 0.68, 0.76, 0.76, 0.76, 0.72, 0.72, 0.76, 0.76,
       0.76, 0.76, 0.76]),
'split1_test_score': array([0.72, 0.8 , 0.72, 0.8 , 0.76, 0.64, 0.68, 0.76, 0.72, 0.72, 0.76,
       0.8 , 0.8 , 0.8 , 0.8 , 0.8 , 0.8 , 0.84, 0.8 , 0.76, 0.8 , 0.84,
       0.8 , 0.76, 0.8 ]),
'split2_test_score': array([0.76, 0.8 , 0.76, 0.72, 0.76, 0.76, 0.76, 0.84, 0.8 , 0.84, 0.8 ,
       0.68, 0.68, 0.68, 0.76, 0.72, 0.72, 0.72, 0.72, 0.72, 0.72, 0.72,
       0.72, 0.72, 0.72]),
'split3_test_score': array([0.68, 0.76, 0.72, 0.76, 0.72, 0.76, 0.72, 0.72, 0.72, 0.64, 0.64,
       0.64, 0.64, 0.64, 0.64, 0.64, 0.64, 0.68, 0.64, 0.64, 0.64, 0.64,
       0.64, 0.64, 0.64]),
'split4_test_score': array([0.58333333, 0.625     , 0.54166667, 0.5       , 0.58333333,
       0.58333333, 0.58333333, 0.54166667, 0.66666667, 0.625     ,
       0.66666667, 0.70833333, 0.75      , 0.79166667, 0.75      ,
       0.75      , 0.75      , 0.75      , 0.75      , 0.75      ,
       0.66666667, 0.70833333, 0.625     , 0.70833333, 0.625     ]),
'std_fit_time': array([1.75434696e-04, 1.14902915e-05, 2.51370532e-05, 2.52122885e-05,
       3.11119346e-05, 2.89635302e-05, 8.93786703e-06, 1.60536278e-05,
       6.28365000e-05, 7.94432172e-05, 1.29494434e-05, 2.07516770e-05,
       9.27249058e-06, 1.44195746e-05, 2.29399810e-05, 2.75950735e-05,
       7.97673175e-06, 2.36077257e-05, 3.79262139e-05, 1.90845696e-05,
       2.11047717e-05, 3.03514998e-05, 2.60830361e-05, 1.47090158e-05,
       1.40117120e-05]),
'std_score_time': array([1.37916341e-03, 6.04260871e-05, 4.13133651e-05, 3.17832623e-05,
       4.14269691e-05, 5.57025404e-05, 1.33542024e-04, 7.21311996e-05,
       2.36597190e-04, 1.20416574e-04, 2.75299033e-05, 7.81933559e-05,
       8.58588437e-05, 2.96332715e-05, 9.45415018e-05, 1.37476494e-04,
       3.37587238e-05, 5.39544142e-05, 1.09812647e-04, 5.60046410e-05,
       6.63293342e-05, 3.01580560e-05, 1.99296257e-05, 3.96460778e-05,
       2.10552629e-05]),
'std_test_score': array([0.06146363, 0.0770714 , 0.08301272, 0.1105622 , 0.07006029,
       0.07111806, 0.06146363, 0.10224372, 0.05491003, 0.08729261,
       0.07495184, 0.05676071, 0.05782733, 0.0621861 , 0.05782733,
       0.05351635, 0.05351635, 0.05291503, 0.052     , 0.04214262,
       0.05866667, 0.06573009, 0.06755738, 0.04406561, 0.06755738])}
```

Лучшая модель:

```
[95] classific_gs.best_estimator_
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=22, p=2,
                     weights='uniform')
```

Лучшее значение метрики:

```
[96] classific_gs.best_score_
```

```
0.7500000000000001
```

Лучшее значение параметров:

```
[97] classific_gs.best_params_
```

```
{'n_neighbors': 22}
```

Вывод: видим, что лучшее найденное значение гиперпараметра К=22. При этом К наилучшее значение метрики = 0,75.