МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
им. Н.Э.Баумана

Отчет по лабораторной работе №3
по курсу «Технологии машинного обучения»

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

Подготовил
Ионов С.А.
ИУ5-62Б

1) Описание задания

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.

2) Текст программы

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
sns.set(style="ticks")
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from typing import Dict, Tuple
from sklearn.metrics import accuracy_score, balanced_accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import plot_roc_curve, plot_precision_recall_curve
from sklearn.model_selection import cross_validate, KFold, RepeatedKFold, ShuffleSplit
from sklearn.model_selection import StratifiedKFold, RepeatedStratifiedKFold,
StratifiedShuffleSplit
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import validation_curve, learning_curve
```

0) Подготовка

```
data = pd.read_csv('data/StudentsPerformance.csv', sep=",")
data[data.duplicated(keep=False)]
data.info()
data.isna().sum()

def show_dependencies(data):
    fig, ax = plt.subplots(2,3,figsize=(15,10))
```

```
    sns.countplot(ax=ax[0][0], x = 'gender', data = data, hue = 'lunch')
    sns.countplot(ax=ax[0][1], x = 'race/ethnicity', data = data, hue = 'lunch')
    sns.countplot(ax=ax[0][2], x = 'test preparation course', data = data, hue = 'lunch')
    sns.countplot(ax=ax[1][0], x = 'parental level of education', data = data, hue = 'lunch')
    sns.scatterplot(ax=ax[1][1], x = 'reading score', y = 'math score', data = data, hue = 'lunch')
    sns.scatterplot(ax=ax[1][2], x = 'writing score', y = 'reading score', data = data, hue = 'lunch')

    plt.setp(ax[1][0].xaxis.get_majorticklabels(), rotation=45)
    plt.show()

show_dependencies(data)

data['gender'].unique()
data['race/ethnicity'].unique()
data['parental level of education'].unique()
data['test preparation course'].unique()
```

2) Множество объектов - все категориальные признаки и math score

```
def get_encoding_features(data):
    data = pd.get_dummies(data, drop_first=True)
    data.rename(columns={'lunch_standard': 'lunch'}, inplace=True)
    return data

dataMath = data.drop(columns=['reading score', 'writing score'])
dataMath = get_encoding_features(dataMath)
sc = MinMaxScaler()
dataMath['math score'] = sc.fit_transform(dataMath[['math score']])

TEST_SIZE = 0.2
RANDOM_STATE = 0

def get_train_test(data):
    data_X = data.drop(columns='lunch')
    data_Y = data['lunch']
    return train_test_split(data_X, data_Y, test_size=TEST_SIZE,
random_state=RANDOM_STATE)

dataMath_X_train, dataMath_X_test, \
dataMath_Y_train, dataMath_Y_test = get_train_test(dataMath)

def class_proportions(array: np.ndarray) -> Dict[int, Tuple[int, float]]:
    """
    Вычисляет пропорции классов
    array - массив, содержащий метки классов
    """
    # Получение меток классов и количества меток каждого класса
    labels, counts = np.unique(array, return_counts=True)
    # Превращаем количество меток в процент их встречаемости
```

```
    # делим количество меток каждого класса на общее количество меток
    counts_perc = counts/array.size
    # Теперь sum(counts_perc)==1.0
    # Создаем результирующий словарь,
    # ключом словаря является метка класса,
    # а значением словаря процент встречаемости метки
    res = dict()
    for label, count2 in zip(labels, zip(counts, counts_perc)):
        res[label] = count2
    return res

def print_class_proportions(array: np.ndarray):
    """
    Вывод пропорций классов
    """
    proportions = class_proportions(array)
    if len(proportions)>0:
        print('Метка \t Количество \t Процент встречаемости')
    for i in proportions:
        val, val_perc = proportions[i]
        val_perc_100 = round(val_perc * 100, 2)
        print('{} \t {} \t \t {}%'.format(i, val, val_perc_100))

print_class_proportions(dataMath['lunch'])

N_DEFAULT = 5 # произвольное K

def print_metrics(X_train, Y_train, X_test, Y_test, criteria='balanced_accuracy',
K=N_DEFAULT):
    ret = 0
    clf = KNeighborsClassifier(n_neighbors=K)
    clf.fit(X_train, Y_train)
    target = clf.predict(X_test)
    if criteria=='balanced_accuracy':
        ret = balanced_accuracy_score(Y_test, target)
        print(f'Сбалансированная оценка: {ret}')
        plot_precision_recall_curve(clf, X_test, Y_test)
    elif criteria=='accuracy':
        ret = accuracy_score(Y_test, target)
        print(f'Обычная оценка: {ret}')
        plot_roc_curve(clf, X_test, Y_test)
    else:
        raise IOError('Неверный критерий. Ожидается "balanced_accuracy" или "accuracy"')
    plt.show()
    print(f'Матрица ошибок:\n {confusion_matrix(Y_test, target)}')
    print(classification_report(Y_test, target, target_names=['free/reduced', 'standard']))
    return ret
```

```
any_res_for_Math = print_metrics(dataMath_X_train, dataMath_Y_train, dataMath_X_test,
dataMath_Y_test)

SPLITS_DEFAULT = 5
N_REP = 2

# перечисляем основные параметры для GridSearchCV
estimator_dis = KNeighborsClassifier()
grid_dis = {"n_neighbors":np.arange(1,50)}

def print_gridResults(grid, strategy):
    print(f'\nСТРАТЕГИЯ {strategy}')
    print(f'Подобранный параметр: {grid.best_params_}')
    print(f'Оценка при подобранном параметре: {grid.best_score_}')
    return [grid.best_params_["n_neighbors"], grid.best_score_]

def get_res_cross_val(data_X, data_Y, scoring_dis='balanced_accuracy'):
    if scoring_dis == 'balanced_accuracy':
        cv_KFold_dis = StratifiedKFold(n_splits=SPLITS_DEFAULT)
        KFold_dis = GridSearchCV(estimator_dis, grid_dis, scoring=scoring_dis,
cv=cv_KFold_dis)
        KFold_dis.fit(data_X, data_Y)
        grid = print_gridResults(KFold_dis, 'Stratified K-fold')
        cv_RepeatedKFold_dis = RepeatedStratifiedKFold(n_splits=SPLITS_DEFAULT,
n_repeats=N_REP, random_state=RANDOM_STATE)
        RepeatedKFold_dis = GridSearchCV(estimator_dis, grid_dis, scoring=scoring_dis,
cv=cv_RepeatedKFold_dis)
        RepeatedKFold_dis.fit(data_X, data_Y)
        grid_rep = print_gridResults(RepeatedKFold_dis, f'Stratified Repeated K-fold: кол-во
поторений: {N_REP}')
    elif scoring_dis == 'accuracy':
        cv_KFold_dis = KFold(n_splits=SPLITS_DEFAULT)
        KFold_dis = GridSearchCV(estimator_dis, grid_dis, scoring=scoring_dis,
cv=cv_KFold_dis)
        KFold_dis.fit(data_X, data_Y)
        grid = print_gridResults(KFold_dis, 'K-fold')
        cv_RepeatedKFold_dis = RepeatedKFold(n_splits=SPLITS_DEFAULT,
n_repeats=N_REP, random_state=RANDOM_STATE)
        RepeatedKFold_dis = GridSearchCV(estimator_dis, grid_dis, scoring=scoring_dis,
cv=cv_RepeatedKFold_dis)
        RepeatedKFold_dis.fit(data_X, data_Y)
        grid_rep = print_gridResults(RepeatedKFold_dis, f'Repeated K-fold: кол-во поторений:
{N_REP}')
    cv_ShuffleSplit_dis = ShuffleSplit(n_splits=SPLITS_DEFAULT,
test_size=1/SPLITS_DEFAULT, random_state=RANDOM_STATE)
    ShuffleSplit_dis = GridSearchCV(estimator_dis, grid_dis, scoring=scoring_dis,
cv=cv_ShuffleSplit_dis)
    ShuffleSplit_dis.fit(data_X, data_Y)
```

```
    grid_shuffle = print_gridResults(ShuffleSplit_dis, f'Shuffle Split: кол-во перемешиваний
{SPLITS_DEFAULT} для ' +
                                     f'тестовой выборки с долей {1/SPLITS_DEFAULT}')
    return grid, grid_rep, grid_shuffle

grid_for_Math, REPgrid_for_Math, SHUFgrid_for_Math = get_res_cross_val(dataMath_X_train,
dataMath_Y_train)

t_grid_for_Math = print_metrics(dataMath_X_train, dataMath_Y_train, dataMath_X_test,
dataMath_Y_test, K=grid_for_Math[0])

t_Repeated_grid_for_Math = print_metrics(dataMath_X_train, dataMath_Y_train,
dataMath_X_test, dataMath_Y_test, \
                                         K=REPgrid_for_Math[0])

t_ShuffleSplit_grid_for_Math = print_metrics(dataMath_X_train, dataMath_Y_train,
dataMath_X_test, dataMath_Y_test, \
                                           K=SHUFgrid_for_Math[0])

def plot_validation_curve(data_X, data_Y):
    param_range = np.arange(1, 100, 5)
    train_scores, test_scores = validation_curve(estimator=estimator_dis, X=data_X, y=data_Y,
                                                 param_name='n_neighbors', param_range=param_range, cv=5)
    train_mean = np.mean(train_scores, axis=1)
    train_std = np.std(train_scores, axis=1)
    test_mean = np.mean(test_scores, axis=1)
    test_std = np.std(test_scores, axis=1)
    plt.figure(figsize=(7,5))
    plt.plot(param_range, train_mean, color='blue', marker='o', markersize=5,
label='тренировочная верность')
    plt.fill_between(param_range, train_mean+train_std, train_mean-train_std, alpha=0.15,
color='blue')
    plt.plot(param_range, test_mean, color='green', linestyle='--', marker='s', markersize=5,
label='проверочная верность')
    plt.fill_between(param_range, test_mean+test_std, test_mean-test_std, alpha=0.15,
color='green')
    plt.grid()
    plt.legend(loc='upper right')
    plt.xlabel('Параметр K')
    plt.ylabel('Верность')
    plt.show()

def plot_learning_curve(data_X, data_Y, n=5):
    train_sizes, train_scores, test_scores =
learning_curve(estimator=KNeighborsClassifier(n_neighbors=n), X=data_X,
                                                y=data_Y,
                                                train_sizes=np.linspace(0.1, 1.0, 10), cv=5)
    train_mean = np.mean(train_scores, axis=1)
```

```
    train_std = np.std(train_scores, axis=1)
    test_mean = np.mean(test_scores, axis=1)
    test_std = np.std(test_scores, axis=1)
    plt.figure(figsize=(7,5))
    plt.plot(train_sizes, train_mean, color='blue', marker='o', markersize=5,
label=f'тренировочная верность, k={n}')
    plt.fill_between(train_sizes, train_mean+train_std, train_mean-train_std, alpha=0.15,
color='blue')
    plt.plot(train_sizes, test_mean, color='green', linestyle='--', marker='s', markersize=5,
             label=f'проверочная верность, k={n}')
    plt.fill_between(train_sizes, test_mean+test_std, test_mean-test_std, alpha=0.15,
color='green')
    plt.grid()
    plt.legend(loc='upper right')
    plt.xlabel('Число тренировочных образцов')
    plt.ylabel('Верность')
    plt.show()

plot_validation_curve(dataMath_X_train, dataMath_Y_train)
plot_learning_curve(dataMath_X_train, dataMath_Y_train, n=2)
plot_learning_curve(dataMath_X_train, dataMath_Y_train, n=4)

from imblearn.under_sampling import RandomUnderSampler
from imblearn.over_sampling import RandomOverSampler

def get_balance(X_train, Y_train, method, strategy):
    if method=='under':
        rus = RandomUnderSampler(random_state=RANDOM_STATE,
sampling_strategy=strategy)
        return rus.fit_resample(X_train, Y_train)
    elif method=='over':
        ros = RandomOverSampler(random_state=RANDOM_STATE,
sampling_strategy=strategy)
        return ros.fit_resample(X_train, Y_train)
    else:
        raise IOError('Неверный метод. Ожидается "under" или "over".')

dataMathUnder_X_train, dataMathUnder_Y_train = get_balance(dataMath_X_train,
dataMath_Y_train, 'under', 1)
print('Результат для произвольного K:')
any_res_for_MathUn = print_metrics(dataMathUnder_X_train, dataMathUnder_Y_train,
dataMath_X_test,
                                   dataMath_Y_test, criteria='accuracy')
grid_for_MathUn, REPgrid_for_MathUn, SHUFgrid_for_MathUn =
get_res_cross_val(dataMathUnder_X_train,
                                                    dataMathUnder_Y_train, 'accuracy')
```

```
dataMathOver_X_train, dataMathOver_Y_train = get_balance(dataMath_X_train,
dataMath_Y_train, 'over', 1)
print('Результат для произвольного K:')
any_res_for_MathOv = print_metrics(dataMathOver_X_train, dataMathOver_Y_train,
dataMath_X_test,
                                   dataMath_Y_test, criteria='accuracy')
grid_for_MathOv, REPgrid_for_MathOv, SHUFgrid_for_MathOv =
get_res_cross_val(dataMathOver_X_train,
                                                        dataMathOver_Y_train, 'accuracy')
```

3) Множество объектов - все категориальные признаки и math score кроме gender и test preparation course

Аналогично

4) Итоги

```
lst_label_cv = ['K-fold', 'Repeated K-fold', 'Shuffle Split']
dc_score = {'Оценка на кросс-валидации':[grid_for_Math[1], REPgrid_for_Math[1],
SHUFgrid_for_Math[1]],\
        'Оценка на отложенной выборке':[t_grid_for_Math, t_Repeated_grid_for_Math, \
                        t_ShuffleSplit_grid_for_Math]}
dc_scoreUn = {'Оценка на кросс-валидации':[grid_for_MathUn[1], REPgrid_for_MathUn[1],
SHUFgrid_for_MathUn[1]],\
        'Оценка на отложенной выборке':[t_grid_for_MathUn, t_Repeated_grid_for_MathUn, \
                        t_ShuffleSplit_grid_for_MathUn]}
dc_scoreOv = {'Оценка на кросс-валидации':[grid_for_MathOv[1], REPgrid_for_MathOv[1],
SHUFgrid_for_MathOv[1]],\
        'Оценка на отложенной выборке':[t_grid_for_MathOv, t_Repeated_grid_for_MathOv, \
                        t_ShuffleSplit_grid_for_MathOv]}
dc_scoreGT = {'Оценка на кросс-валидации':[grid_for_MathGT[1], REPgrid_for_MathGT[1],
SHUFgrid_for_MathGT[1]],\
        'Оценка на отложенной выборке':[t_grid_for_MathGT, t_Repeated_grid_for_MathGT, \
                        t_ShuffleSplit_grid_for_MathGT]}
dc_scoreGTUn = {'Оценка на кросс-валидации':[grid_for_MathGTUn[1],
REPgrid_for_MathGTUn[1], SHUFgrid_for_MathGTUn[1]],\
        'Оценка на отложенной выборке':[t_grid_for_MathGTUn,
t_Repeated_grid_for_MathGTUn, \
                        t_ShuffleSplit_grid_for_MathGTUn]}

print(f'Accuracy: {any_res_for_Math} при k = {N_DEFAULT}')
print('Accuracy для кросс-валидации:')
pd.DataFrame(dc_score, index=lst_label_cv)

print(f'Accuracy: {any_res_for_MathUn} при k = {N_DEFAULT}')
print('Accuracy для кросс-валидации:')
pd.DataFrame(dc_scoreUn, index=lst_label_cv)

print(f'Accuracy: {any_res_for_MathOv} при k = {N_DEFAULT}')
print('Accuracy для кросс-валидации:')
```

```
pd.DataFrame(dc_scoreOv, index=lst_label_cv)

print(f'Accuracy: {any_res_for_MathGT} при k = {N_DEFAULT}')
print('Accuracy для кросс-валидации:')
pd.DataFrame(dc_scoreGT, index=lst_label_cv)

print(f'Accuracy: {any_res_for_MathGTUn} при k = {N_DEFAULT}')
print('Accuracy для кросс-валидации:')
pd.DataFrame(dc_scoreGTUn, index=lst_label_cv)
```

3) Экранные формы с примерами выполнения программы

1. Подготовка

- Для лабораторной работы был выбран Students Performance in Exams, в котором поставлена задача классификации с целевым признаком «тип обеда» («lunch»)
- Первые 5 строк набора данных имеют вид:

	gender	race/ethnicity	parental level of education	lunch	test preparation course	math score	reading score	writing score
0	female	group B	bachelor's degree	standard	none	72	72	74
1	female	group C	some college	standard	completed	69	90	88
2	female	group B	master's degree	standard	none	90	95	93
3	male	group A	associate's degree	free/reduced	none	47	57	44
4	male	group C	some college	standard	none	76	78	75

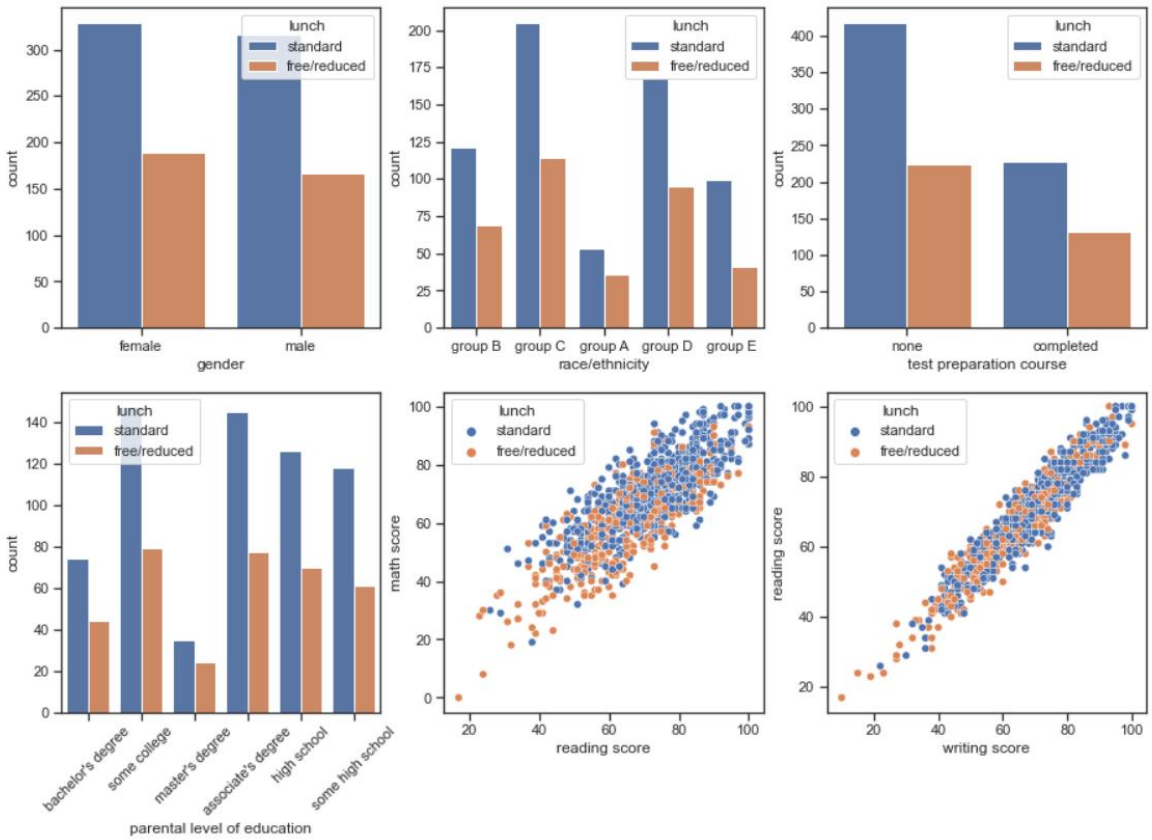
- В наборе данных отсутствуют пропуски:

```
gender                       0
race/ethnicity               0
parental level of education  0
lunch                        0
test preparation course      0
math score                   0
reading score                0
writing score                0
dtype: int64
```

- Состав признаков в датасете:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 8 columns):
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
 0   gender                       1000 non-null   object
 1   race/ethnicity               1000 non-null   object
 2   parental level of education  1000 non-null   object
 3   lunch                        1000 non-null   object
 4   test preparation course      1000 non-null   object
 5   math score                   1000 non-null   int64
 6   reading score                1000 non-null   int64
 7   writing score                1000 non-null   int64
dtypes: int64(3), object(5)
memory usage: 62.6+ KB
```

- Исследование зависимостей между целевым и остальными признаками:



Из диаграмм видно, что на тип обеда мало влияют «gender» и «test preparation course». В дальнейшем мы избавимся от этих признаков, чтобы повысить accuracy. Также из выполнения первой лабораторной работы известно, что между количественными признаками в данном датасете существует сильная корреляция, поэтому было принято решение оставить один из количественных столбцов: «math score».

- Далее была произведена проверка наличия случайных ошибок в категориях датасета:

```
: data['gender'].unique()
: array(['female', 'male'], dtype=object)

: data['race/ethnicity'].unique()
: array(['group B', 'group C', 'group A', 'group D', 'group E'],
      dtype=object)

: data['parental level of education'].unique()
: array(["bachelor's degree", 'some college', "master's degree",
       "associate's degree", 'high school', 'some high school'],
      dtype=object)

: data['test preparation course'].unique()
: array(['none', 'completed'], dtype=object)
```

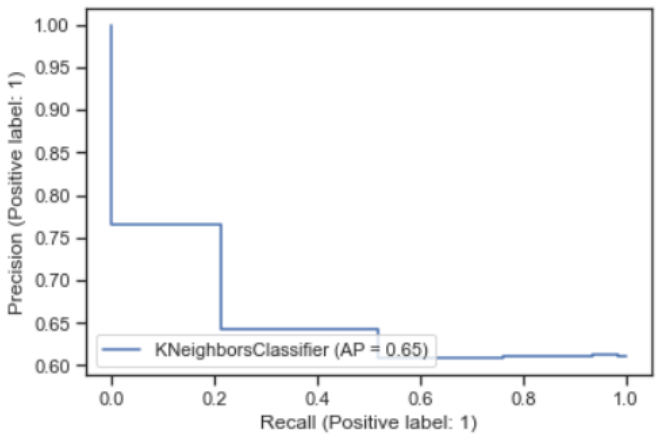
- Далее для всех категориальных признаков был применен метод «one-hot encoding» для того, чтобы закодировать их числами, и исключить появления отношения порядка между объектами
- Было замечено, что у нас имеются несбалансированные признаки (класс «standard» количественно превышает класс «free/reduced»). Хотя меньший класс составляет больше 10% относительно большего, в дальнейшем были предприняты попытки устранения этой несбалансированности и получения более высоких accuracy:

Метка	Количество	Процент встречаемости
0	355	35.5%
1	645	64.5%

2. **Применение метода ближайших соседей к датасету со всеми категориальными признаками и «math score»**

- Выборка была поделена на обучающую (на ней будет выполнен подбор гиперпараметра K) и отложенную (на ней будет замерена итоговая accuracy при полученном гиперпараметре K после его подбора с помощью решетчатого поиска и кросс-валидации) в отношении 80% против 20%
- При обучении с произвольным K (K=5) были получены следующие значения метрик:

```
Сбалансированная оценка: 0.4965321563682219
```



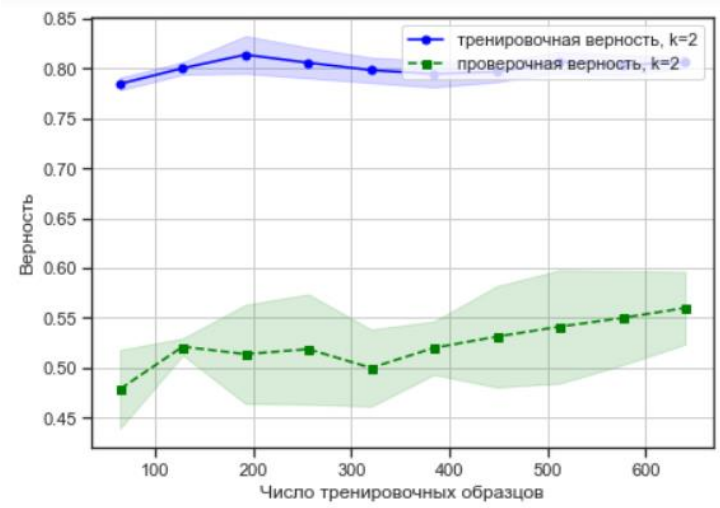

```
Матрица ошибок:
 [[18 60]
 [29 93]]
              precision    recall  f1-score   support

free/reduced       0.38      0.23      0.29        78
    standard       0.61      0.76      0.68       122

    accuracy                           0.56       200
   macro avg       0.50      0.50      0.48       200
weighted avg       0.52      0.56      0.52       200
```

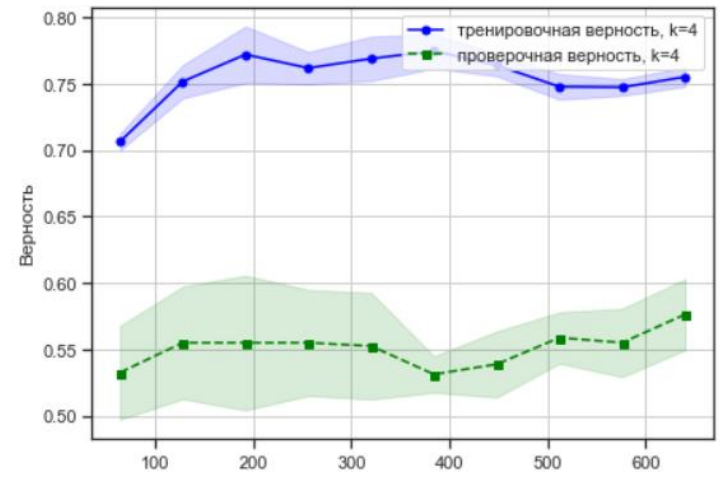
Отсюда видно, что результаты предсказания класса «standard» значительно лучше, чем результаты предсказания класса «free/reduced».

- При использовании кросс-валидации были получены следующие оценки:

```
СТРАТЕГИЯ Stratified K-fold
Подобранный параметр: {'n_neighbors': 2}
Оценка при подобранном параметре: 0.5771003996003995

СТРАТЕГИЯ Stratified Repeated K-fold: кол-во поторений: 2
Подобранный параметр: {'n_neighbors': 4}
Оценка при подобранном параметре: 0.5568448218448219

СТРАТЕГИЯ Shuffle Split: кол-во перемешиваний 5 для тестовой выборки с долей 0.2
Подобранный параметр: {'n_neighbors': 2}
Оценка при подобранном параметре: 0.5943368508534446
```

- В дальнейшем после проведения проверки на отложенной выборке было установлено, что стратегия Repeated K-fold дает более лучшую accuracy, чем это делает K-fold. Это видно также по кривым обучения:





3. Балансирование обучающего набора

- После применения метода «undersampling», то есть случайное уменьшение количества объектов преобладающего класса, были получены следующие метрики оценки:

```
СТРАТЕГИЯ K-fold
Подобранный параметр: {'n_neighbors': 2}
Оценка при подобранном параметре: 0.44178542178542185

СТРАТЕГИЯ Repeated K-fold: кол-во поторений: 2
Подобранный параметр: {'n_neighbors': 48}
Оценка при подобранном параметре: 0.5795167895167894

СТРАТЕГИЯ Shuffle Split: кол-во перемешиваний 5 для тестовой выборки с долей 0.2
Подобранный параметр: {'n_neighbors': 33}
Оценка при подобранном параметре: 0.5747747747747748
```

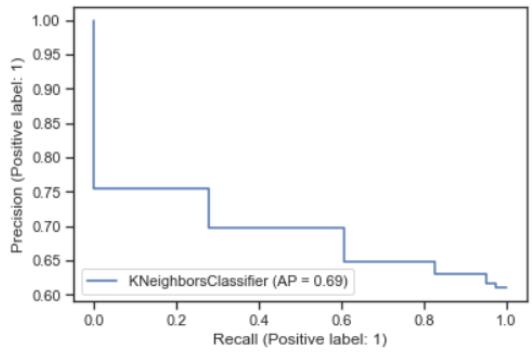
- После применения метода «oversampling», то есть случайное дублирование объектов с принадлежностью меньшему классу, были получены следующие метрики оценки:

```
СТРАТЕГИЯ K-fold
Подобранный параметр: {'n_neighbors': 1}
Оценка при подобранном параметре: 0.7486352244246981

СТРАТЕГИЯ Repeated K-fold: кол-во поторений: 2
Подобранный параметр: {'n_neighbors': 1}
Оценка при подобранном параметре: 0.7309022556390976

СТРАТЕГИЯ Shuffle Split: кол-во перемешиваний 5 для тестовой выборки с долей 0.2
Подобранный параметр: {'n_neighbors': 1}
Оценка при подобранном параметре: 0.7142857142857144
```

4. Применение метода ближайших соседей к датасету с отброшенными категориальными признаками «gender» и «test preparation course»

- В этом случае даже при произвольном K видны улучшения в метриках оценивания:

```
Сбалансированная оценка: 0.5613703236654056
```




```
Матрица ошибок:
 [[ 23  55]
 [ 21 101]]
              precision    recall  f1-score   support

free/reduced       0.52      0.29      0.38        78
    standard       0.65      0.83      0.73       122

    accuracy                           0.62       200
   macro avg       0.59      0.56      0.55       200
weighted avg       0.60      0.62      0.59       200
```

- Далее к данной выборке были применены все те шаги, что были рассмотрены в предыдущих пунктах

5. **Итоги**

3.1. Все категории и math score

При произвольном K

```
print(f'Accuracy: {any_res_for_Math} при k = {N_DEFAULT}')
```

```
Accuracy: 0.4965321563682219 при k = 5
```

С кросс-валидацией и GridSearch

```
print('Accuracy для кросс-валидации:')
pd.DataFrame(dc_score, index=lst_label_cv)
```

```
Accuracy для кросс-валидации:
```

	Оценка на кросс-валидации	Оценка на отложенной выборке
K-fold	0.577100	0.516183
Repeated K-fold	0.556845	0.529424
Shuffle Split	0.594337	0.516183

3.2. Все категории и math_score с undersampling

При произвольном K

```
print(f'Accuracy: {any_res_for_MathUn} при k = {N_DEFAULT}')
```

```
Accuracy: 0.515 при k = 5
```

С кросс-валидацией и GridSearch

```
print('Accuracy для кросс-валидации:')
pd.DataFrame(dc_scoreUn, index=lst_label_cv)
```

```
Accuracy для кросс-валидации:
```

	Оценка на кросс-валидации	Оценка на отложенной выборке
K-fold	0.441785	0.475
Repeated K-fold	0.579517	0.590
Shuffle Split	0.574775	0.650

3.3. Все категории и math score с oversampling

При произвольном K

```
print(f'Accuracy: {any_res_for_MathOv} при k = {N_DEFAULT}')
```

```
Accuracy: 0.5 при k = 5
```

С кросс-валидацией и GridSearch

```
print('Accuracy для кросс-валидации:')
pd.DataFrame(dc_scoreOv, index=lst_label_cv)
```

```
Accuracy для кросс-валидации:
```

	Оценка на кросс-валидации	Оценка на отложенной выборке
K-fold	0.748635	0.535
Repeated K-fold	0.730902	0.545
Shuffle Split	0.714286	0.545

3.4. Без признаков Gender и Test preparation course

При произвольном K

```
print(f'Accuracy: {any_res_for_MathGT} при k = {N_DEFAULT}')
```

```
Accuracy: 0.5613703236654056 при k = 5
```

С кросс-валидацией и GridSearch

```
print('Accuracy для кросс-валидации:')
pd.DataFrame(dc_scoreGT, index=lst_label_cv)
```

```
Accuracy для кросс-валидации:
```

	Оценка на кросс-валидации	Оценка на отложенной выборке
K-fold	0.602413	0.612757
Repeated K-fold	0.580231	0.629361
Shuffle Split	0.580119	0.629361

3.5. Без признаков Gender и Test preparation course с undersampling

При произвольном K

```
print(f'Accuracy: {any_res_for_MathGTUn} при k = {N_DEFAULT}')
```

```
Accuracy: 0.5860655737704918 при k = 5
```

С кросс-валидацией и GridSearch

```
print('Accuracy для кросс-валидации:')
pd.DataFrame(dc_scoreGTUn, index=lst_label_cv)
```

```
Accuracy для кросс-валидации:
```

	Оценка на кросс-валидации	Оценка на отложенной выборке
K-fold	0.485029	0.575
Repeated K-fold	0.580303	0.645
Shuffle Split	0.587387	0.645