In [ ]:
```
surname = "Марчук"  # Ваша фамилия

alph = 'абвгдеёжзийклмнопрстуфхцчшщъыьэюя'
w = [4, 42, 21, 21, 55,  1, 44, 26, 18, 3, 38, 26, 18, 12,  3, 49, 45,
       7, 42, 9,  4,  3, 36, 33, 31, 29,  5, 4,  4, 19, 21, 27, 33]
d = dict(zip(alph, w))
variant =  sum([d[el] for el in surname.lower()]) % 40 + 1

print("Задание № 2. Вариант: ", variant % 2 + 1)
print("Задание № 3. Вариант: ", variant % 3 + 1)
```

```
Задание № 2. Вариант:  2
Задание № 3. Вариант:  3
```

In [ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

1. Реализация собственных классов и функций

In [ ]:
```
# Загрузка данных из файла
data = pd.read_csv("regularization.csv")

data
```

Out[ ]:

	X1	X2	X3	X4	X5	X6	X7	
0	3.856603	14.873388	57.360757	221.217682	853.148822	3290.256492	1.268921e+04	4.893726
1	0.103760	0.010766	0.001117	0.000116	0.000012	0.000001	1.294799e-07	1.34348(
2	3.168241	10.037752	31.802020	100.756468	319.220791	1011.368453	3.204259e+03	1.015187
3	3.744019	14.017681	52.482471	196.495391	735.682558	2754.409777	1.031256e+04	3.861044
4	2.492535	6.212731	15.485450	38.598027	96.206935	239.799159	5.977078e+02	1.489808
...	...	...	...	...	...	...	...	
295	4.403960	19.394866	85.414221	376.160841	1656.597410	7295.589233	3.212949e+04	1.414970
296	3.004771	9.028649	27.129023	81.516502	244.938425	735.983886	2.211463e+03	6.644940
297	3.226139	10.407971	33.577559	108.325862	349.474260	1127.452444	3.637318e+03	1.173449
298	0.283141	0.080169	0.022699	0.006427	0.001820	0.000515	1.458880e-04	4.13068
299	1.487420	2.212420	3.290798	4.894801	7.280627	10.829354	1.610780e+01	2.395908

300 rows × 17 columns

In [ ]:
```
# Преобразование данных в питру массив
xData = np.array(data.iloc[:, :-1].values.tolist())
yData = np.array(data.iloc[:, -1].values.tolist())
# Посмотрим полученные массивы
xData
```

Out[ ]:
```
array([[3.85660322e+00, 1.48733884e+01, 5.73607574e+01, ...,
        1.61016146e+08, 6.20975387e+08, 2.39485567e+09],
       [1.03759747e-01, 1.07660851e-02, 1.11708626e-03, ...,
        1.67650420e-14, 1.73953651e-15, 1.80493868e-16],
       [3.16824117e+00, 1.00377521e+01, 3.18020196e+01, ...,
        1.02672769e+07, 3.25292093e+07, 1.03060380e+08],
       ...,
       [3.22613873e+00, 1.04079711e+01, 3.35775586e+01, ...,
        1.32300822e+07, 4.26820805e+07, 1.37698313e+08],
       [2.83141014e-01, 8.01688339e-02, 2.26990849e-02, ...,
        2.12833000e-08, 6.02617515e-09, 1.70625734e-09],
       [1.48742048e+00, 2.21241970e+00, 3.29079838e+00, ...,
        2.59461316e+02, 3.85928077e+02, 5.74037327e+02]])
```

In [ ]:
```
yData
```

Out[ ]:
```
array([ 9.20975909, 10.40924034,  7.64374171,  8.45334146,  9.31782425,
        9.88279025, 10.40431337,  8.55347846, 10.28258081, 10.11235376,
        8.57232874, 10.73153928,  9.52733591,  9.44350604,  8.88600518,
        8.69602507,  7.82066865, 10.14869949, 10.45958369,  7.83724452,
        7.77761255,  9.39765763,  9.8608622 ,  9.55117601,  9.90363916,
       10.2243049 ,  8.50730601,  9.62307336,  8.24828071,  8.8179605 ,
        8.59075044,  8.60236282,  9.6839511 ,  9.74167235, 10.85764473,
       10.14805827, 10.44002588,  8.71500186, 10.74226863,  8.06837814,
        9.07117575,  8.64123208,  9.80121151,  8.86596983,  9.43386002,
        8.37104231,  9.89765928,  9.51844691, 10.23640952, 10.2061975 ,
        9.61228127, 10.35314762,  8.55938649, 10.62908238, 10.09203451,
       10.12283434, 11.74999269,  9.71451446,  8.80896182, 10.01296735,
        8.4083225 ,  9.3046388 ,  9.37218162,  8.20230789, 10.05575522,
        9.54474432, 10.70656231,  9.32392779,  9.75036952,  8.49430575,
       10.25758897,  9.6450204 ,  9.88908331,  8.90463669, 10.01862541,
       11.33907708,  8.56888434,  8.82016768,  8.78291415,  9.08531274,
        9.48837458,  9.79464733,  7.59464072, 10.05696088, 10.01701388,
        8.99713201, 10.01760063, 10.41009948,  8.80097655,  7.65818585,
        9.69774064, 10.20030521, 10.12145389, 10.06525583, 10.24778617,
        9.48626086, 10.17947131,  9.39286191, 10.39713512,  7.90990402,
        8.65865507,  8.13171061,  9.69901529,  9.44169302,  8.52188139,
        8.18338362,  8.41959346, 10.51524985, 10.19432725,  8.27238304,
       10.20048498,  8.69317059,  9.56121299,  9.90874866,  9.96578181,
        9.72589703, 10.01216067, 10.35048931,  9.51759511,  8.95940156,
        9.96452016,  7.80964664, 10.84804753,  9.67703435,  9.86720296,
        8.69434212, 10.78426231, 10.36207141,  9.22133284, 10.45193906,
       10.08509652,  8.20239328,  9.91992195,  9.84228875,  9.15409772,
        9.69130817,  9.31969654,  8.82816113, 11.02974044, 11.15430562,
        8.47800644,  7.96590317,  8.64957771,  9.05777287,  8.7637843 ,
        9.17680816,  9.71199737,  9.22630804,  8.52385103, 10.79406034,
        9.61034186, 10.48725327,  9.63102388,  9.06169077, 10.08428646,
        8.72176342,  8.30358526,  8.82784528, 10.74278134,  7.88578998,
       10.6765041 , 10.06398312, 10.30625523,  9.60601848,  7.29761035,
        8.09572614,  9.49900111,  9.88464657,  9.09383004,  8.26092173,
        7.74236795,  9.34575003,  9.05406912,  8.8718252 ,  8.10730302,
        9.92218723,  8.73720739, 10.58194168,  9.74605925,  8.77694901,
        8.38102397,  8.66619538, 10.54739232, 10.40847648, 10.15424971,
        8.49723072,  9.37575245,  9.39919161, 10.04064996, 10.28734285,
       10.387372  ,  8.27538355,  9.63872697, 10.20364254,  8.42307935,
       10.70211992,  8.8496155 ,  8.2030597 , 10.42183887,  9.922578  ,
        8.8428445 ,  9.21022859,  9.10796714,  9.25855111,  9.10613359,
        9.20976338,  9.38244733,  8.43541287,  8.54404117,  7.60106465,
       10.05705304, 10.15220806, 10.48395054,  9.80235701,  8.85207313,
       10.06793663,  9.28900393, 10.79168924, 10.53006892, 10.75957953,
        9.89824828, 10.09857478,  9.59053866,  9.49240562,  8.82840248,
        9.72195038,  9.77316756,  8.9303852 , 10.35387203,  8.35094711,
        9.3139144 ,  8.95550734,  9.12147426,  8.91365672,  8.73307438,
       10.47510031,  9.56233436,  8.98755549,  8.56397341,  9.99350234,
        9.04344242,  9.87392275,  7.93281796,  8.70997426,  9.77510931,
        8.84364712,  9.00365103,  9.29420849, 11.40594944, 10.81867275,
        8.40051931,  9.61005427,  8.2326173 ,  8.72512689,  8.8645034 ,
       10.79911538,  9.79123429,  8.78528511,  9.9353131 ,  7.94433612,
       11.11648041, 10.39018905,  9.55150546,  9.85800537,  7.99499988,
       10.31667054,  9.82144422, 10.42201791,  9.34555316,  8.48691883,
       10.35409192,  8.5208006 ,  9.78349713,  9.69807216,  9.82213449,
        9.83985011,  9.03045085,  9.48687604,  9.01538514, 10.11435195,
        8.12688591, 10.06230777,  9.94798209, 10.11137557, 10.83044608,
        9.75978156,  9.94364694, 10.14625269, 10.04366355,  9.06111763,
        9.76842763,  8.4220947 , 10.28844464,  8.66963872,  9.85165223,
        9.51003239,  8.61531979,  8.32016008, 10.21788567, 10.47669189])
```

1.1 Реализуйте класс, предназначенный для оценки параметров линейной регрессии с регуляризацией совместимый с sklearn.

Передаваемые параметры:

1. коэффициент регуляризации (alpha).

Использовать метод наименьших квадратов с регуляризацией.

In [ ]:
```
# реализованный класс:

class RegularizedLinearRegression:
    def __init__(self, alpha=1.0):
        self.alpha = alpha
        self.weights = None

    def fit(self, X, y):
        # Добавляем столбец с единицами для учёта свободного члена
        X = np.column_stack((np.ones(len(X)), X))

        # Рассчитываем вектор весов w = (X^T*X + alpha*I)^-1 * X^T * y
        identity = np.identity(X.shape[1])
        self.weights = np.linalg.inv(X.T @ X + self.alpha * identity) @ X.T @ y

    def predict(self, X):
        # Добавляем столбец с единицами для учёта свободного члена
        X = np.column_stack((np.ones(len(X)), X))

        # Предсказываем значения y
        y_pred = X @ self.weights
        return y_pred
```

In [ ]:
```
# Создаем экземпляр класса с коэффициентом регуляризации alpha = 0.1
model = RegularizedLinearRegression(alpha=0.1)

# Обучаем модель на данных xData и yData
model.fit(xData, yData)

# Предсказываем значения y для тех же данных xData
y_pred = model.predict(xData)
```

In [ ]:
```
# y_pred будет содержать предсказанные значения целевой переменной
#  на основе данных xData, обученных с использованием регуляризированной линейной р

y_pred
```

Out[ ]:
```
array([ 9.86985872e+00, -6.32353893e+01,  1.08815703e+01,  1.00682733e+01,
        9.96521887e+00, -1.65828467e+01, -2.17705986e+01,  9.96261697e+00,
       -2.78716214e+01, -4.68605335e+01,  1.06042237e+01,  9.91503257e+00,
       -6.70290493e+01,  1.02509850e+01,  9.61500576e+00,  1.09338525e+01,
        1.03130264e+01, -5.65459299e+00,  9.98861002e+00,  1.03751866e+01,
        1.06823582e+01, -3.39409500e+01,  3.49803128e+00,  1.06797435e+01,
        8.00485464e+00,  7.61980256e+00,  1.09249074e+01,  1.02684661e+01,
        1.08125632e+01,  1.09429550e+01,  9.64385599e+00,  1.04133092e+01,
        9.96505944e+00, -2.08473190e+00, -4.63398789e+01, -4.44710414e+00,
       -4.06089474e+01,  9.58595413e+00, -5.70274531e+01,  1.09045446e+01,
        1.07307226e+01,  9.59712036e+00, -2.15918144e+01,  9.65339103e+00,
        1.48947640e+00,  1.00152174e+01, -5.08883321e+00,  9.82327718e+00,
       -1.33795846e+00, -2.87715415e+01,  5.03124946e+00, -4.56035407e+01,
        9.59365048e+00, -3.18799356e+01,  4.38634401e+00,  9.94093226e+00,
        1.08055066e+01,  8.63876968e+00,  9.58731963e+00, -1.18862875e+01,
        1.09425693e+01,  9.93983521e+00,  1.05931389e+01,  1.09367819e+01,
       -5.88638765e+01,  2.03003632e+00, -4.90078049e+01, -3.81801751e+00,
       -7.38429126e-01,  9.84924478e+00, -5.87012707e+01,  7.38295138e+00,
       -2.61298740e+00,  1.08714442e+01,  9.52123832e-01, -5.79459674e+01,
        9.79547441e+00,  9.93879104e+00,  9.78292894e+00,  6.70332658e+00,
        1.09411657e+01,  1.02744685e+01,  1.09427117e+01, -1.01075159e+01,
       -4.42420044e+00, -6.21545335e+01, -4.13291763e+00, -1.34755910e+01,
        1.08097579e+01,  1.08578283e+01,  9.32058923e+00, -5.52592504e+00,
       -5.27887954e+01,  1.02725139e+01,  2.53684352e-01,  9.88280233e+00,
        1.02043023e+01,  7.95447703e+00, -2.20172250e+00,  1.03836724e+01,
        1.09105359e+01,  9.63984270e+00, -5.18432727e+01,  8.95575804e+00,
        9.78587336e+00,  1.03405470e+01,  1.09301430e+01, -5.93877604e+01,
        1.38997269e+00,  1.08450489e+01, -4.57724005e+00,  1.02536251e+01,
        1.06839845e+01, -3.00656195e+01, -5.61578695e+01,  4.93255815e-02,
       -4.20406585e+01, -2.57555566e+01,  9.83625986e+00,  2.79408514e+00,
       -1.77128701e+01,  1.00346813e+01, -4.23280242e+01,  1.01066653e+01,
        9.14257543e+00,  1.09428165e+01, -3.26861581e+01, -2.46674859e+01,
        1.08368853e+01, -5.66073998e+01, -1.23522035e+01,  1.06809721e+01,
       -1.62173917e+01,  4.17015470e+00,  9.99206022e+00,  9.99121629e+00,
        1.08642702e+01,  1.05797424e+01, -6.45873899e+01,  1.02351573e+01,
        1.08925435e+01,  9.71775370e+00,  1.08354199e+01,  9.77769850e+00,
        1.09264460e+01,  1.04233268e+01, -3.24771531e+01,  7.32690126e+00,
        1.05419008e+01, -4.28772711e+01,  7.89825936e+00, -2.85099611e+01,
        1.01494868e+01,  9.59764786e+00, -4.64264652e+01,  9.66910571e+00,
        1.08557977e+01,  1.09356277e+01, -2.17629515e+01,  7.72645198e+00,
       -5.09677659e+00, -5.92769558e+01, -6.09118587e+01,  8.50715206e+00,
        1.01042786e+01,  1.08078127e+01,  4.46513987e+00, -2.80989274e+01,
        9.59218018e+00,  1.09429546e+01,  9.78019411e+00,  9.61834687e+00,
        1.09424226e+01,  9.79253269e+00,  1.09287087e+01, -6.31397784e+01,
        1.00531396e+01, -2.63684069e+01,  8.72691557e+00,  1.02681813e+01,
        9.59055752e+00,  9.60546423e+00, -2.66329053e+01, -6.46386195e+01,
        9.61924363e+00,  1.01243164e+01,  8.65460976e+00,  6.66662230e+00,
       -3.99482039e+01,  1.42863008e-01, -4.53097507e+01,  1.03644562e+01,
       -4.96303428e+01, -2.01934625e+01,  1.08954489e+01, -5.38236916e+00,
        1.07861250e+01,  9.65152252e+00,  1.44569562e+00, -4.56084437e+01,
        9.61277057e+00,  9.76454036e+00,  5.09698487e+00,  9.69460357e+00,
        4.37971734e+00, -1.09035933e+01,  9.58579910e+00,  1.01818064e+01,
        1.01620666e+01,  1.08401185e+01, -1.87804038e+01,  9.90056032e+00,
        9.95930913e+00, -2.35707987e+00,  9.70099007e+00, -3.19634593e+00,
       -6.21862883e+01, -5.64696545e+01, -2.45429227e+01, -5.16129740e+01,
       -1.08753549e+01,  9.98043553e+00,  8.70063459e+00, -3.67361798e+01,
        9.62458765e+00,  5.80732648e+00, -6.23804115e+01,  9.65329726e+00,
       -8.22075275e+00,  1.04212033e+01,  1.89411718e+00,  9.70184623e+00,
        9.60610949e+00,  1.06352128e+01,  1.08198587e+01, -3.85639713e+01,
        3.86121122e+00,  7.34292130e+00,  1.02325072e+01,  9.87586665e+00,
       -4.51267499e+00,  5.31680115e+00,  9.70820391e+00,  6.38219114e+00,
       -2.44923327e+01,  9.59570465e+00,  1.08764681e+01, -5.38196587e+01,
        1.09160446e+01, -1.58485894e+01,  1.09304847e+01,  3.03344491e+00,
        9.65534308e+00,  1.09009114e+01,  9.59345904e+00, -6.16592038e+00,
```

```
-8.14477414e+00,  1.09429534e+01,  9.96172523e+00,  1.04361436e+01,
-9.91757551e+00, -3.86055047e+01,  8.73517778e+00,  1.01296746e+01,
 1.06373490e+01, -2.27506902e+01, -5.70655184e+01,  9.91620135e+00,
 9.66239039e+00,  9.58623504e+00,  9.94056426e+00,  1.09429395e+01,
 9.98335730e+00, -3.28813308e+01, -4.06147459e+01,  3.38497993e+00,
 9.91004849e+00, -1.85511104e+00,  1.08656631e+01, -6.58600379e+01,
 1.06916359e+01,  4.26710895e-02,  9.91417646e+00,  9.95395286e+00,
 4.38692984e+00,  5.29348982e+00,  9.90987886e+00, -5.41205037e+01,
 5.93569452e+00, -3.67447347e+01, -4.71973939e+01,  9.77104877e+00,
-3.93401026e+00,  1.09261421e+01,  9.78547298e+00,  9.80153987e+00,
 1.09429649e+01,  1.08362050e+01, -5.46575115e+01, -4.88096457e+00])
```

1.2 Реализуйте класс для стандартизации признаков в виде трансформации совместимый с sklearn. Передаваемые параметры:

1. has_bias (содержит ли матрица вектор единиц),

2. apply_mean (производить ли центровку)

In [ ]:
```python
# Реализация класса:
# Параметры
# - has_bias: указывает, содержит ли матрица вектор единиц (по умолчанию True).
# - apply_mean: указывает, следует ли применять центрирование (по умолчанию True).
# - alpha: параметр для незначительной регуляризации (по умолчанию 1e-10).


from sklearn.base import TransformerMixin

class CustomStandardScaler(TransformerMixin):
    def __init__(self, has_bias=True, apply_mean=True, alpha=1e-10):
        self.has_bias = has_bias
        self.apply_mean = apply_mean
        self.alpha = alpha
        self.mean_ = None
        self.std_ = None

    def fit(self, X, y=None):
        if self.has_bias:
            X = X[:, 1:]  # Исключаем столбец с единицами, если он есть

        if self.apply_mean:
            self.mean_ = np.mean(X, axis=0)
            X -= self.mean_

        self.std_ = np.std(X, axis=0)
        self.std_[self.std_ == 0] = 1  # Предотвращаем деление на ноль
        return self

    def transform(self, X):
        if self.has_bias:
            X = X[:, 1:]  # Исключаем столбец с единицами, если он есть

        if self.apply_mean:
            X -= self.mean_

        X /= self.std_

        if self.has_bias:
            X = np.column_stack((np.ones(len(X)), X))  # Добавляем столбец с едини

        return X
```

In [ ]:
```
# Создаем экземпляр класса с параметрами
scaler = CustomStandardScaler(has_bias=True, apply_mean=True)

# Обучаем scaler на данных xData
scaler.fit(xData)

# Преобразуем данные xData с использованием обученного scaler
X_transformed = scaler.transform(xData)

# Вывод
X_transformed
```

Out[ ]:
```
array([[ 1.        , -0.1367899 , -0.02256199, ..., -0.53250246,
        -0.53973979, -0.54272836],
       [ 1.        , -2.21925631, -1.74755458, ..., -0.72187453,
        -0.6913591 , -0.66388666],
       [ 1.        , -0.81433184, -0.79119709, ..., -0.70979912,
        -0.68341667, -0.65867273],
       ...,
       [ 1.        , -0.76245886, -0.73780081, ..., -0.70631455,
        -0.68093771, -0.65692036],
       [ 1.        , -2.20953199, -1.74690553, ..., -0.72187453,
        -0.6913591 , -0.66388666],
       [ 1.        , -1.91077308, -1.64862307, ..., -0.72187422,
        -0.69135901, -0.66388663]])
```

1.3 Реализуйте функции для расчета MSE и R^2 при отложенной выборке (run_holdout) и кросс-валидации (run_cross_val). Для кросс-валидации используйте только класс KFold. Выходными значениями должны быть MSE и R^2 для обучающей и тестовой частей.

In [ ]:
```
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import train_test_split, KFold

def run_holdout(model, X, y, train_size, random_state=0) -> dict:
    # Разбиваем данные на обучающую и тестовую выборки
    X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=train_size

    # Обучаем модель на обучающей выборке
    model.fit(X_train, y_train)

    # Предсказываем значения на обучающей и тестовой выборках
    y_train_pred = model.predict(X_train)
    y_test_pred = model.predict(X_test)

    # Рассчитываем MSE и R^2 для обучающей и тестовой выборок
    mse_train = mean_squared_error(y_train, y_train_pred)
    mse_test = mean_squared_error(y_test, y_test_pred)
    r2_train = r2_score(y_train, y_train_pred)
    r2_test = r2_score(y_test, y_test_pred)

    # Возвращаем словарь с результатами
    scores = {'MSE_train': mse_train, 'MSE_test': mse_test, 'R2_train': r2_train, '
    return scores

def run_cross_val(model, X, y, n_splits, shuffle=True, random_state=0) -> dict:
    # Создаем объект KFold для кросс-валидации
    kf = KFold(n_splits=n_splits, shuffle=shuffle, random_state=random_state)

    # Списки для хранения результатов
    mse_train_list = []
    mse_test_list = []
```

```
    r2_train_list = []
    r2_test_list = []

    # Цикл по фолдам
    for train_index, test_index in kf.split(X):
        # Разбиваем данные на обучающую и тестовую выборки
        X_train, X_test = X[train_index], X[test_index]
        y_train, y_test = y[train_index], y[test_index]

        # Обучаем модель на обучающей выборке
        model.fit(X_train, y_train)

        # Предсказываем значения на обучающей и тестовой выборках
        y_train_pred = model.predict(X_train)
        y_test_pred = model.predict(X_test)

        # Рассчитываем MSE и R^2 для обучающей и тестовой выборок
        mse_train = mean_squared_error(y_train, y_train_pred)
        mse_test = mean_squared_error(y_test, y_test_pred)
        r2_train = r2_score(y_train, y_train_pred)
        r2_test = r2_score(y_test, y_test_pred)

        # Добавляем результаты в списки
        mse_train_list.append(mse_train)
        mse_test_list.append(mse_test)
        r2_train_list.append(r2_train)
        r2_test_list.append(r2_test)

    # Средние значения MSE и R^2 по всем фолдам
    mse_train_mean = np.mean(mse_train_list)
    mse_test_mean = np.mean(mse_test_list)
    r2_train_mean = np.mean(r2_train_list)
    r2_test_mean = np.mean(r2_test_list)

    # Возвращаем словарь с результатами
    scores = {'MSE_train': mse_train_mean, 'MSE_test': mse_test_mean, 'R2_train': r
    return scores
```

In [ ]:
```
# Пример использования
# Передаём модель обученную на шаге 1.1

# параметр train_size в функции run_holdout определяет размер обучающей
# выборки при использовании метода отложенной выборки (holdout).
# Здесь 80 процентов данных используются для обучения

run_holdout(model, xData, yData, train_size = 0.8)
```

Out[ ]:
```
{'MSE_train': 0.22864613680553572,
 'MSE_test': 0.22483738366300252,
 'R2_train': 0.6779671591941744,
 'R2_test': 0.6949480257819375}
```

In [ ]:
```
# Параметр n_splits в функции run_cross_val указывает на количество фолдов (разбиен
# которые будут использоваться при кросс-валидации.
# Кросс-валидация помогает оценить производительность модели путем разделения данны
# на несколько частей (фолдов), обучения модели на некоторых из них и оценки ее на

run_cross_val(model, xData, yData, n_splits=5)
```

Out[ ]:
```
{'MSE_train': 0.22528931963948326,
 'MSE_test': 0.23521306384164356,
 'R2_train': 0.6877547667306956,
 'R2_test': 0.6702532112838864}
```

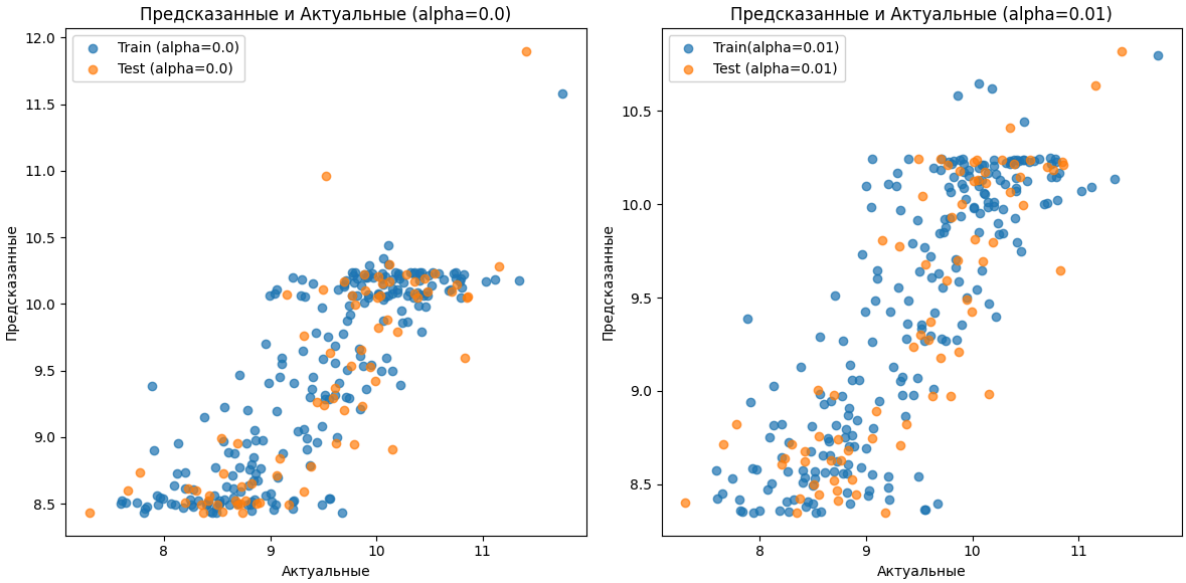
1.4 Используя класс Pipeline, выполнить обучение линейной регрессии с предварительной стандартизацией с коэффициентом регуляризации равным 0 и 0.01. Выведите значения параметров обученной модели. Выведите значения MSE и R^2, полученные посредством функций run_holdout и run_cross_val. Отобразите график предсказание (y^) - действительное значение (y) для разных коэффициентов регуляризации для обучающего и текстового множества. Использовать следующие параметры:

train_size=0.75, n_splits=4, shuffle=True, random_state=0

In [ ]:
```
from sklearn.pipeline import Pipeline
from sklearn.linear_model import Ridge
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import train_test_split, cross_val_predict

# Создание конвейера для обучения линейной регрессии с предварительной стандартизац
pipeline_0 = Pipeline([
    ('scaler', StandardScaler()),  # Предварительная стандартизация признаков
    ('ridge', Ridge(alpha=0.0))  # Линейная регрессия без регуляризации
])

pipeline_001 = Pipeline([
    ('scaler', StandardScaler()),  # Предварительная стандартизация признаков
    ('ridge', Ridge(alpha=0.01))  # Линейная регрессия с регуляризацией (alpha = 0.
])

# Разделение данных на обучающий и тестовый наборы
X_train, X_test, y_train, y_test = train_test_split(xData, yData, train_size=0.75,

# Обучение моделей
pipeline_0.fit(X_train, y_train)
pipeline_001.fit(X_train, y_train)

# Предсказание значений на обучающем и тестовом наборах для разных значений alpha
y_pred_train_0 = pipeline_0.predict(X_train)
y_pred_test_0 = pipeline_0.predict(X_test)

y_pred_train_001 = pipeline_001.predict(X_train)
y_pred_test_001 = pipeline_001.predict(X_test)

# Вывод параметров моделей
print("Параметры модели (alpha=0.0):", pipeline_0.named_steps['ridge'].coef_)
print("Параметры модели (alpha=0.01):", pipeline_001.named_steps['ridge'].coef_)

# Вычисление MSE и R^2 для обучающего и тестового наборов данных
mse_train_0 = mean_squared_error(y_train, y_pred_train_0)
mse_test_0 = mean_squared_error(y_test, y_pred_test_0)
r2_train_0 = r2_score(y_train, y_pred_train_0)
r2_test_0 = r2_score(y_test, y_pred_test_0)

mse_train_001 = mean_squared_error(y_train, y_pred_train_001)
mse_test_001 = mean_squared_error(y_test, y_pred_test_001)
r2_train_001 = r2_score(y_train, y_pred_train_001)
r2_test_001 = r2_score(y_test, y_pred_test_001)

print("\nЗначения для alpha=0.0:")
print("  MSE (train):", mse_train_0)
print("  MSE (test):", mse_test_0)
print("  R^2 (train):", r2_train_0)
```

```python
print("  R^2 (test):", r2_test_0)

print("\nЗначения для alpha=0.01:")
print("  MSE (train):", mse_train_001)
print("  MSE (test):", mse_test_001)
print("  R^2 (train):", r2_train_001)
print("  R^2 (test):", r2_test_001)

# Построение графиков
plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
plt.scatter(y_train, y_pred_train_0, label='Train (alpha=0.0)', alpha=0.7)
plt.scatter(y_test, y_pred_test_0, label='Test (alpha=0.0)', alpha=0.7)
plt.xlabel('Актуальные')
plt.ylabel('Предсказанные')
plt.title('Предсказанные и Актуальные (alpha=0.0)')
plt.legend()

plt.subplot(1, 2, 2)
plt.scatter(y_train, y_pred_train_001, label='Train(alpha=0.01)', alpha=0.7)
plt.scatter(y_test, y_pred_test_001, label='Test (alpha=0.01)', alpha=0.7)
plt.xlabel('Актуальные')
plt.ylabel('Предсказанные')
plt.title('Предсказанные и Актуальные (alpha=0.01)')
plt.legend()

plt.tight_layout()
plt.show()
```

```
Параметры модели (alpha=0.0): [-5.60009641e+01  2.77172721e+03 -6.40865535e+04  8.
63829553e+05
 -7.53659105e+06  4.52648874e+07 -1.94439583e+08  6.11584602e+08
 -1.42702480e+09  2.48066982e+09 -3.19861018e+09  3.01446388e+09
 -2.01605641e+09  9.05750269e+08 -2.45019005e+08  3.01506516e+07]
Параметры модели (alpha=0.01): [ 0.79654942 -2.38952139 -1.75875062  0.71958827
1.72879851  1.38469504
  0.55674554 -0.15302367 -0.51458453 -0.53919318 -0.34670145 -0.07966578
  0.1407045   0.23126325  0.14847729 -0.1171596 ]

Значения для alpha=0.0:
  MSE (train): 0.21418546620079723
  MSE (test): 0.26038748955091917
  R^2 (train): 0.6992088274906549
  R^2 (test): 0.6527274174216351

Значения для alpha=0.01:
  MSE (train): 0.2270305798213734
  MSE (test): 0.22622187296343468
  R^2 (train): 0.6811698033893339
  R^2 (test): 0.6982932851527643
```



Задача 2. Классификация и кросс-валидация (2 балла)

In [ ]:
```
print("Задание № 2. Вариант: ", variant % 2 + 1)
```

```
Задание № 2. Вариант:  2
```

In [ ]:
```
# Загрузка данных из файла
data2 = pd.read_csv("Cl_A5_V2.csv")

data2
```

Out[ ]:

	X1	X2	Y
0	5.712051	4.420663	0
1	4.658783	6.312037	1
2	4.211528	4.934160	0
3	5.440266	5.688972	0
4	5.109973	7.006561	1
...	...	...	...
495	4.782801	5.331527	0
496	3.469108	5.801888	1
497	6.357797	4.195166	1
498	5.261725	4.757229	0
499	5.393892	4.049974	0

500 rows × 3 columns

In [ ]:
```
# Преобразование данных в питру массив
xData = np.array(data2.iloc[:, :-1].values.tolist())
yData = np.array(data2.iloc[:, -1].values.tolist())
# Посмотрим один из полученных массивов
yData
```

Out[ ]:
```
array([0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1,
       1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0,
       0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1,
       1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0,
       0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1,
       1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1,
       1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1,
       0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0,
       1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1,
       1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1,
       1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1,
       0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1,
       0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0,
       0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1,
       1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0,
       0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0,
       0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1,
       1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0,
       1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0,
       0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1,
       0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1,
       0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0])
```

Дано множество наблюдений (см. набор данных к заданию), классификатор - логистическая регрессия. Найти степень полинома с минимальной ошибкой на проверочном подмножестве. Для лучшего случая рассчитать ошибку на тестовом подмножестве. В качестве метрики использовать долю правильных классификаций. Сделать заключение о влиянии степени полинома на качество предсказания.

In [ ]:
```python
from sklearn.preprocessing import PolynomialFeatures
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

# Функция для создания полиномиальных признаков заданной степени
def create_polynomial_features(X, degree):
    poly = PolynomialFeatures(degree=degree)
    return poly.fit_transform(X)

# Разделение данных на обучающий, проверочный и тестовый наборы
X_train, X_temp, y_train, y_temp = train_test_split(xData, yData, test_size=0.3, ra
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, rand

best_degree = None
best_accuracy = 0

# Значения для графика
train_accuracies = []
val_accuracies = []

# Перебор степеней полинома от 1 до 10
for degree in range(1, 11):
    # Создание полиномиальных признаков
    X_train_poly = create_polynomial_features(X_train, degree)
    X_val_poly = create_polynomial_features(X_val, degree)

    # Обучение логистической регрессии
    model = LogisticRegression(penalty='l2', fit_intercept=True,
                               max_iter=100, C=1e5, solver='liblinear', random_stat
    model.fit(X_train_poly, y_train)
```

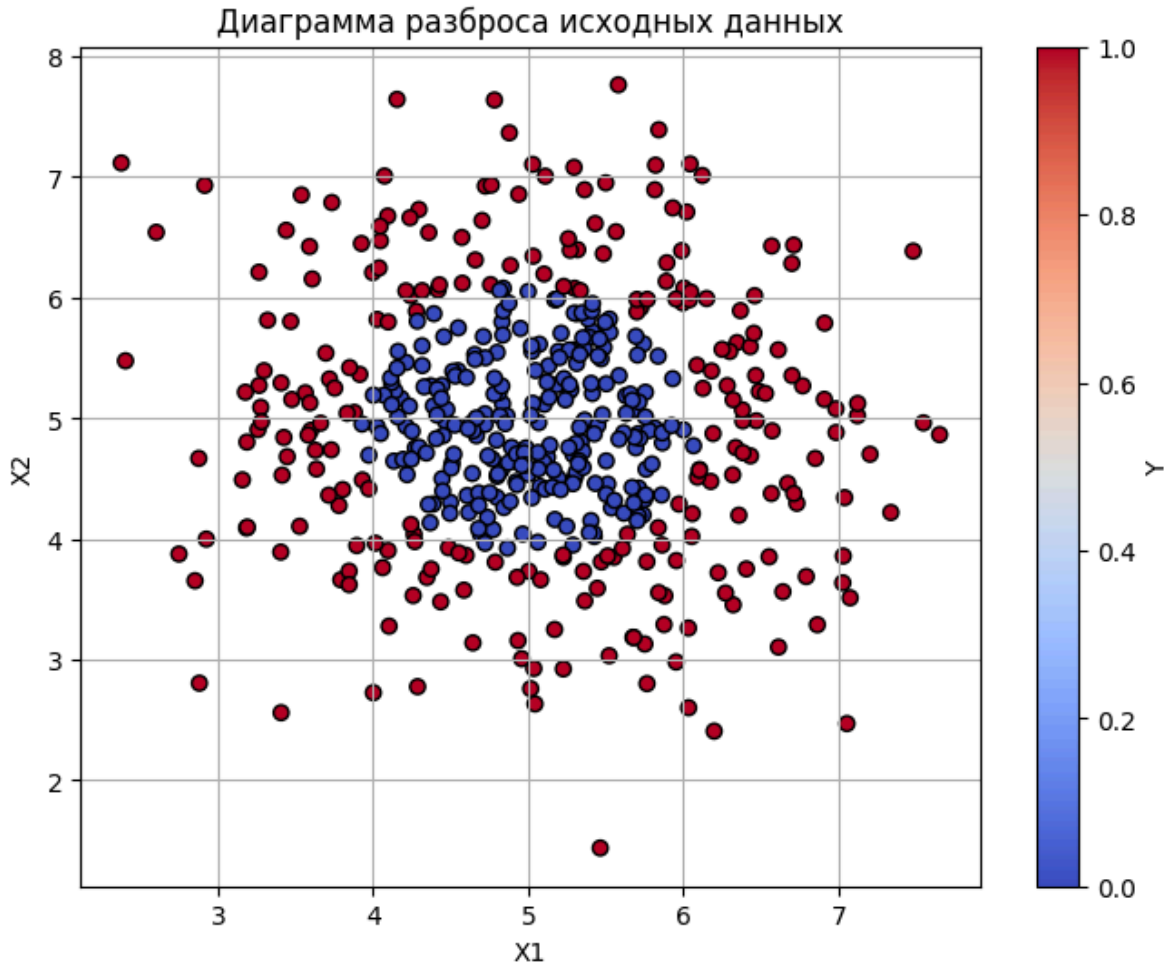
```
    # Предсказание на обучающем и проверочном наборах (для графика)
    train_accuracies.append(accuracy_score(y_train, model.predict(X_train_poly)))
    val_accuracies.append(accuracy_score(y_val, model.predict(X_val_poly)))

    # Предсказание на проверочном наборе
    y_val_pred = model.predict(X_val_poly)
    # Оценка качества предсказания
    accuracy = accuracy_score(y_val, y_val_pred)
    # Если текущая степень полинома дает лучшую точность, обновляем лучшую степень
    if accuracy > best_accuracy:
        best_accuracy = accuracy
        best_degree = degree

# Создание полиномиальных признаков с лучшей степенью для обучающего и тестового на
X_train_best = create_polynomial_features(X_train, best_degree)
X_test_best = create_polynomial_features(X_test, best_degree)

# Обучение логистической регрессии с лучшей степенью
final_model = LogisticRegression(penalty='l2', fit_intercept=True, max_iter=100, C=
final_model.fit(X_train_best, y_train)

# Предсказание на тестовом наборе
y_test_pred = final_model.predict(X_test_best)

# Оценка качества предсказания на тестовом наборе
test_accuracy = accuracy_score(y_test, y_test_pred)

print("Лучшая степень полинома:", best_degree)
print("Точность на тестовом наборе:", test_accuracy)
```

```
Лучшая степень полинома: 2
Точность на тестовом наборе: 1.0
```

Построение графиков:

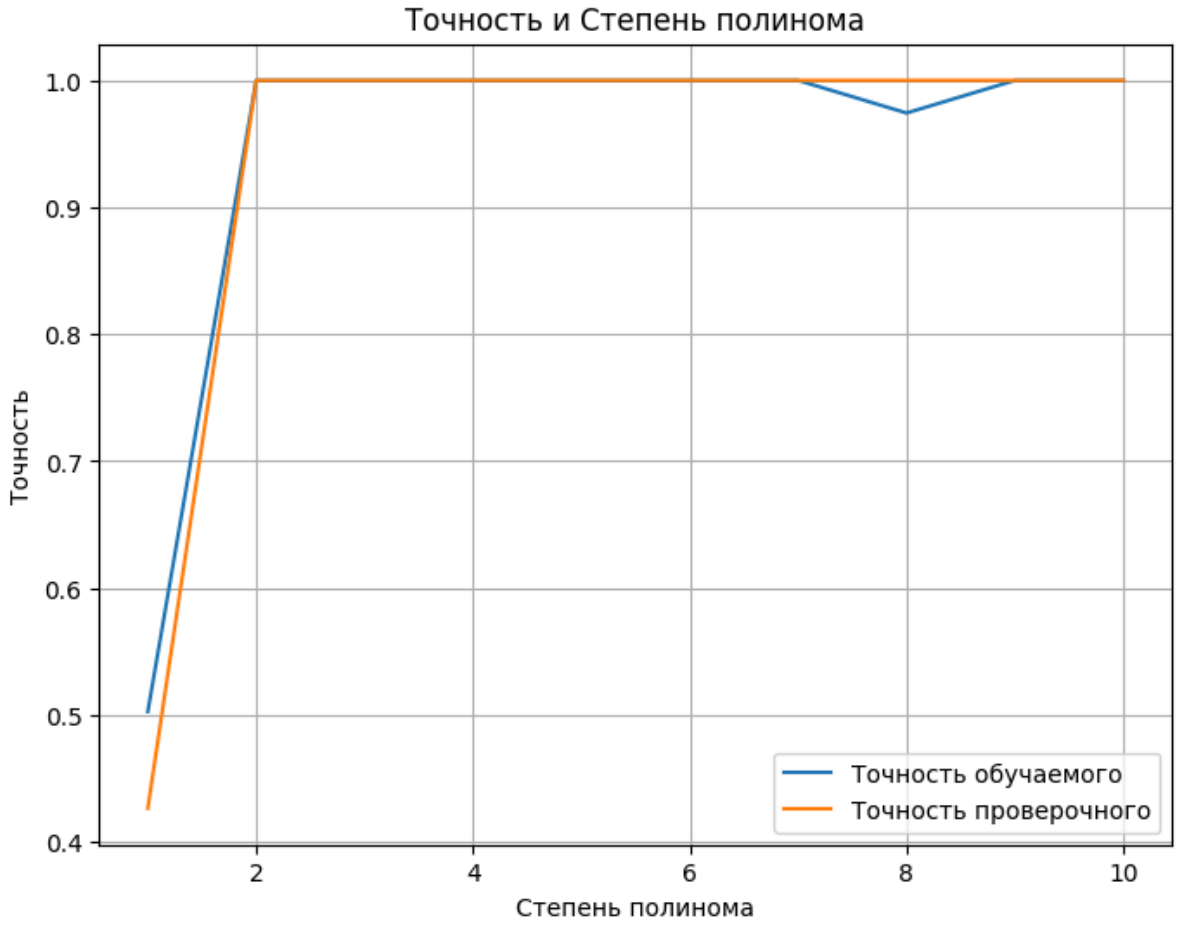
Диаграмма разброса исходных данных:

In [ ]:
```
plt.figure(figsize=(8, 6))
plt.scatter(xData[:, 0], xData[:, 1], c=yData, cmap='coolwarm', edgecolors='k')
plt.xlabel('X1')
plt.ylabel('X2')
plt.title('Диаграмма разброса исходных данных')
plt.colorbar(label='Y')
plt.grid(True)
plt.show()
```



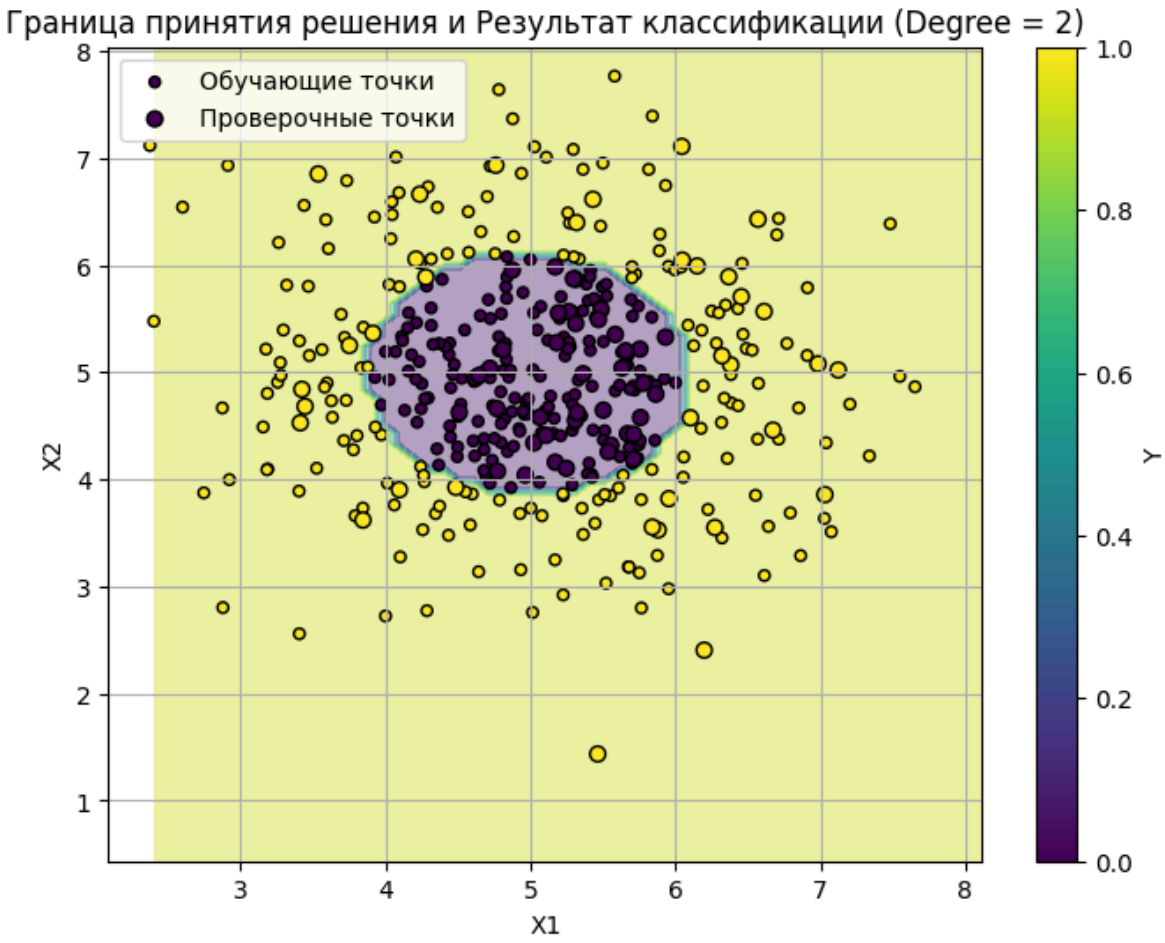
Зависимость доли правильных классификаций от степени полинома для обучающего и проверочного подмножеств:

In [ ]:
```
# Построение графика
degrees = list(range(1, 11))
plt.figure(figsize=(8, 6))
plt.plot(degrees, train_accuracies, label='Точность обучаемого')
plt.plot(degrees, val_accuracies, label='Точность проверочного')
plt.xlabel('Степень полинома')
plt.ylabel('Точность')
plt.title('Точность и Степень полинома')
plt.legend()
plt.grid(True)
plt.show()
```



Результат классификации для наилучшего случая (степень полинома) для обучающего и тестового подмножеств с указанием границы принятия решения:

In [ ]:
```
# Предсказание на обучающем наборе для наилучшей степени полинома
y_train_pred = final_model.predict(X_train_best)

# Построение графика
plt.figure(figsize=(8, 6))

# Отрисовка границы принятия решения
x_min, x_max = X_test[:, 0].min() - 1, X_test[:, 0].max() + 1
y_min, y_max = X_test[:, 1].min() - 1, X_test[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.1), np.arange(y_min, y_max, 0.1))
Z = final_model.predict(create_polynomial_features(np.c_[xx.ravel(), yy.ravel()], b
Z = Z.reshape(xx.shape)
plt.contourf(xx, yy, Z, alpha=0.4)

# Отрисовка обучающих точек
plt.scatter(X_train[:, 0], X_train[:, 1], c=y_train, cmap='viridis', s=20,
            edgecolors='k', label='Обучающие точки')
# Отрисовка проверочных точек
plt.scatter(X_test[:, 0], X_test[:, 1], c=y_test, cmap='viridis', s=40,
            edgecolors='k', label='Проверочные точки')

plt.xlabel('X1')
plt.ylabel('X2')
plt.title('Граница принятия решения и Результат классификации (Degree = {})'.format
plt.colorbar(label='Y')
plt.legend()
plt.grid(True)
plt.show()
```



Можно сделать следующие выводы: В данном случае лучшая степень полинома равна 2.

Модель достигает 100% точности на тестовом наборе данных, что означает, что она идеально классифицирует все точки в этом наборе.

Граница принятия решения, описанная кругом в пределах области X1:[4,6] и X2:[4,6], соответствует области, в которой сосредоточено большинство точек данных. Это может указывать на то, что модель правильно выявила область, где наиболее вероятно нахождение одного из классов.

Учитывая высокую точность на тестовом наборе данных, можно заключить, что модель с полиномом второй степени хорошо обобщает данные. Это означает, что она может успешно классифицировать новые, ранее не виденные данные, с высокой точностью.

Таким образом, вывод можно сделать следующий: модель с полиномом второй степени демонстрирует высокое качество предсказания и хорошую обобщающую способность на данном наборе данных.

Задача 3. Классификация текстовых документов (4 балла)

In [ ]:
```
print("Задание № 3. Вариант: ", variant % 3 + 1 )
```

```
Задание № 3. Вариант:  3
```

3.1 Загрузите исходные данные

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

# Импорт файла reviews.tsv
data = pd.read_csv("reviews.tsv", sep="\t", header=None, names=["label", "text"])

data
```

Out[ ]:

	label	text
0	0	unless bob crane is someone of particular inte...
1	1	finds a way to tell a simple story , perhaps t...
2	0	ill-considered , unholy hokum .
3	0	nijinsky says , 'i know how to suffer' and if ...
4	1	the auteur's ear for the way fears and slights...
...	...	...
10657	0	it's mildly sentimental , unabashedly consumer...
10658	0	so verbally flatfooted and so emotionally pred...
10659	0	alternative medicine obviously has its merits ...
10660	0	a by-the-numbers patient/doctor pic that cover...
10661	0	according to the script , grant and bullock's ...

10662 rows × 2 columns

3.2 Разбейте исходные данные на обучающее (train, 80%) и тестовое подмножества (test, 20%)

In [ ]:
```
# Разделение данных на обучающее и тестовое подмножества
train_data, test_data = train_test_split(data, test_size=0.2, stratify=data["label"
```

3.3 Используя стратифицированную кросс-валидацию k-folds (k=4) для обучающего множество с метрикой Balanced-Accuracy, найти лучшие гиперпараметры для следующих классификаторов:

1. K-ближайших соседей: количество соседей (n) из диапазона np.arange(1, 150, 20)
2. Логистическая регрессия: параметр регуляризации (C) из диапазона np.logspace(-2, 10, 8, base=10)
3. Наивный Байес: сглаживающий параметр модели Бернулли (a) из диапазона np.logspace(-4, 1, 8, base=10)
4. Наивный Байес: сглаживающий параметр полиномиальной модели (a) из диапазона np.logspace(-4, 1, 8, base=10)

In [ ]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import BernoulliNB, MultinomialNB
from sklearn.metrics import make_scorer, balanced_accuracy_score
from sklearn.pipeline import Pipeline
from sklearn.feature_extraction.text import CountVectorizer

# Функция для выполнения кросс-валидации и поиска лучших параметров с использование
```

```
def find_best_params_with_vectorizer(classifier, param_grid, X_train, y_train, ngra
    scorer = make_scorer(balanced_accuracy_score)

    # Создание пайплайна, который сначала преобразует текст в матрицу счетчиков, а
    pipeline = Pipeline([
        ('vectorizer', CountVectorizer(ngram_range = ngram_range)),
        ('classifier', classifier)
    ])

    grid_search = GridSearchCV(pipeline, param_grid, cv=4, scoring=scorer)
    grid_search.fit(X_train, y_train)
    return grid_search.best_params_


def find_best_params_for_classificators(ngram_range):

  # K-ближайших соседей
  knn_param_grid = {'classifier__n_neighbors': np.arange(1, 150, 20)}
  best_knn_params = find_best_params_with_vectorizer(KNeighborsClassifier(), knn_pa
  print("Лучшие параметры для K-ближайших соседей:", best_knn_params)

  # Логистическая регрессия
  log_reg_param_grid = {'classifier__C': np.logspace(-2, 10, 8, base=10)}
  best_log_reg_params = find_best_params_with_vectorizer(LogisticRegression(max_ite
  print("Лучшие параметры для Логистической регрессии:", best_log_reg_params)

  # Наивный Байес с моделью Бернулли
  bernoulli_nb_param_grid = {'classifier__alpha': np.logspace(-4, 1, 8, base=10)}
  best_bernoulli_nb_params = find_best_params_with_vectorizer(BernoulliNB(), bernou
  print("Лучшие параметры для Наивного Байеса (модель Бернулли):", best_bernoulli_r

  # Наивный Байес с полиномиальной моделью
  multinomial_nb_param_grid = {'classifier__alpha': np.logspace(-4, 1, 8, base=10)}
  best_multinomial_nb_params = find_best_params_with_vectorizer(MultinomialNB(), mu
  print("Лучшие параметры для Наивного Байеса (полиномиальная модель):", best_multi


# Запуск функций
#ngram_range=(1, 1)
#find_best_params_for_classificators(ngram_range)
```

Результаты работы в пункте 3.7

3.4 Отобразите кривые (параметры модели)-(Balanced-Accuracy) при обучении и проверке для каждой классификатора (две кривые на одном графике для каждого классификатора)

In [ ]:
```
from sklearn.model_selection import learning_curve

# Функция для отображения кривых обучения
def plot_learning_curve(estimator, title, X, y, ylim=None, cv=None,
                        n_jobs=None, train_sizes=np.linspace(.1, 1.0, 5)):
    plt.figure()
    plt.title(title)
    if ylim is not None:
        plt.ylim(*ylim)
    plt.xlabel("Training examples")
    plt.ylabel("Score")
    train_sizes, train_scores, test_scores = learning_curve(
        estimator, X, y, cv=cv, n_jobs=n_jobs, train_sizes=train_sizes, scoring='ba
    train_scores_mean = np.mean(train_scores, axis=1)
    train_scores_std = np.std(train_scores, axis=1)
```

```
    test_scores_mean = np.mean(test_scores, axis=1)
    test_scores_std = np.std(test_scores, axis=1)
    plt.grid()

    plt.fill_between(train_sizes, train_scores_mean - train_scores_std,
                     train_scores_mean + train_scores_std, alpha=0.1,
                     color="r")
    plt.fill_between(train_sizes, test_scores_mean - test_scores_std,
                     test_scores_mean + test_scores_std, alpha=0.1, color="g")
    plt.plot(train_sizes, train_scores_mean, 'o-', color="r",
             label="Training score")
    plt.plot(train_sizes, test_scores_mean, 'o-', color="g",
             label="Cross-validation score")

    plt.legend(loc="best")
    return plt


def plot_all_curves(ngram):
  # Отображение кривых обучения для каждого классификатора
  classifiers = {
      "К-ближайших соседей": KNeighborsClassifier(n_neighbors=best_knn_params.get('
      "Логистическая регрессия": LogisticRegression(max_iter=1000, C=best_log_reg_p
      "Наивный Байес (модель Бернулли)": BernoulliNB(alpha=best_bernoulli_nb_params
      "Наивный Байес (полиномиальная модель)": MultinomialNB(alpha=best_multinomial
  }

  # Создание пайплайна для каждого классификатора
  pipelines = {}
  for clf_name, clf in classifiers.items():
      pipelines[clf_name] = Pipeline([
          ('vectorizer', CountVectorizer(ngram_range=ngram)),
          ('classifier', clf)
      ])

  # Отображение кривых обучения
  for clf_name, pipeline in pipelines.items():
      plot_learning_curve(pipeline, clf_name, train_data["text"], train_data["label

  plt.show()

# Запуск функций
#ngram_range=(1, 1)
#plot_all_curves(ngram_range)
```

Результаты работы в пункте 3.7

3.5 Если необходимо, выбранные модели обучите на всём обучающем подмножестве (train) и протестируйте на тестовом (test) по Balanced-Accuracy, R, P, F1. Определите время обучения и предсказания.

In [ ]:
```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
import time

# Функция для обучения и тестирования модели
def train_test_model(estimator, X_train, y_train, X_test, y_test):
    # Обучение модели
    start_time = time.time()
    estimator.fit(X_train, y_train)
    train_time = time.time() - start_time

    # Предсказание на тестовом наборе
```

```
    start_time = time.time()
    y_pred = estimator.predict(X_test)
    predict_time = time.time() - start_time

    # Вычисление метрик
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)

    return accuracy, precision, recall, f1, train_time, predict_time


def train_models():

  # Обучение и тестирование моделей для каждого классификатора
  results = {}
  for clf_name, pipeline in pipelines.items():
      # Обучение и тестирование модели
      accuracy, precision, recall, f1, train_time, predict_time = train_test_model(

      # Сохранение результатов
      results[clf_name] = {
          'Accuracy': accuracy,
          'Precision': precision,
          'Recall': recall,
          'F1': f1,
          'Training Time': train_time,
          'Prediction Time': predict_time
      }

  # Вывод результатов
  for clf_name, result in results.items():
      print(f"Модель: {clf_name}")
      print(f"Accuracy: {result['Accuracy']:.4f}")
      print(f"Precision: {result['Precision']:.4f}")
      print(f"Recall: {result['Recall']:.4f}")
      print(f"F1: {result['F1']:.4f}")
      print(f"Training Time: {result['Training Time']:.4f} сек")
      print(f"Prediction Time: {result['Prediction Time']:.4f} сек")
      print()

  return results


# Запуск функций
#train_models()
```

Результаты работы в пункте 3.7

3.6 Выполните пункты 3-5 для n-gram=1, n-gram=2 и n-gram=(1,2)

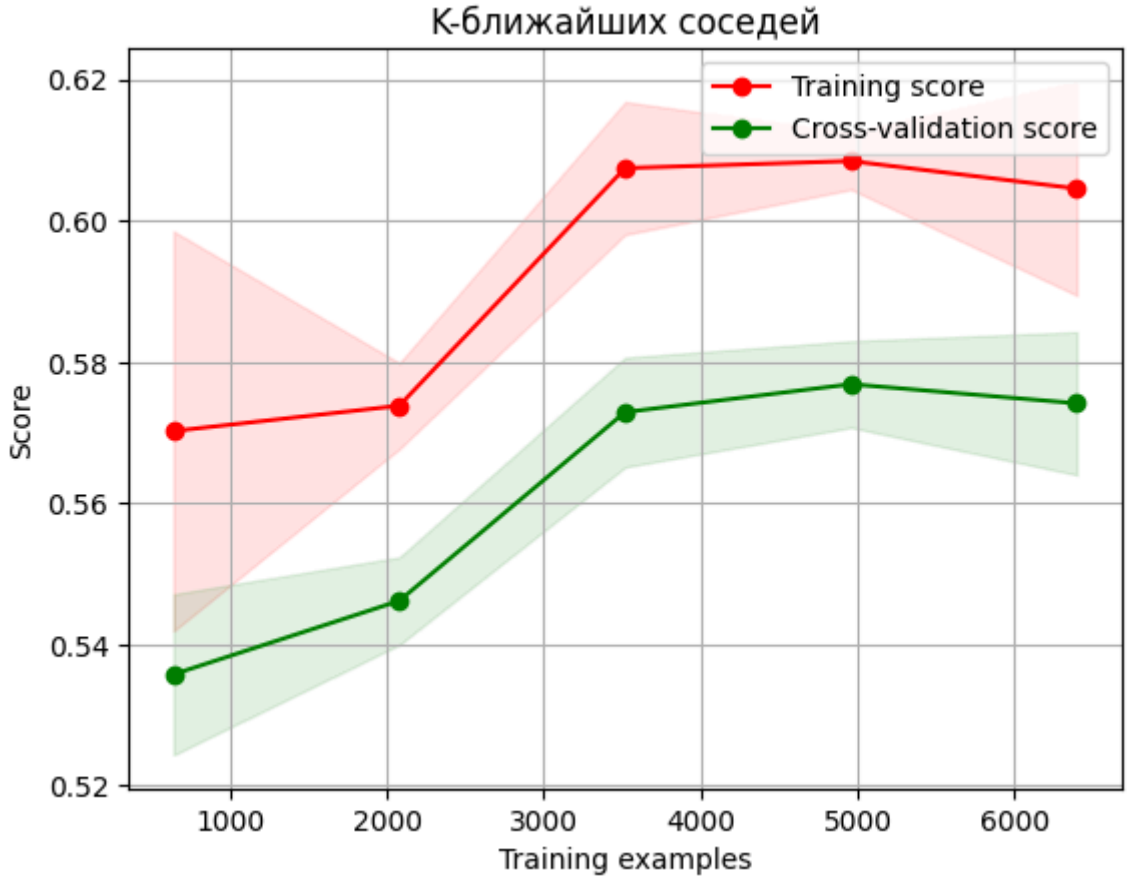
3.7 Выведите в виде таблицы итоговые данные по всем методам для лучших моделей (метод, n-gram, значение параметра модели, время обучения, время предсказания, метрики (Balanced-Accuracy, R, P, F1))

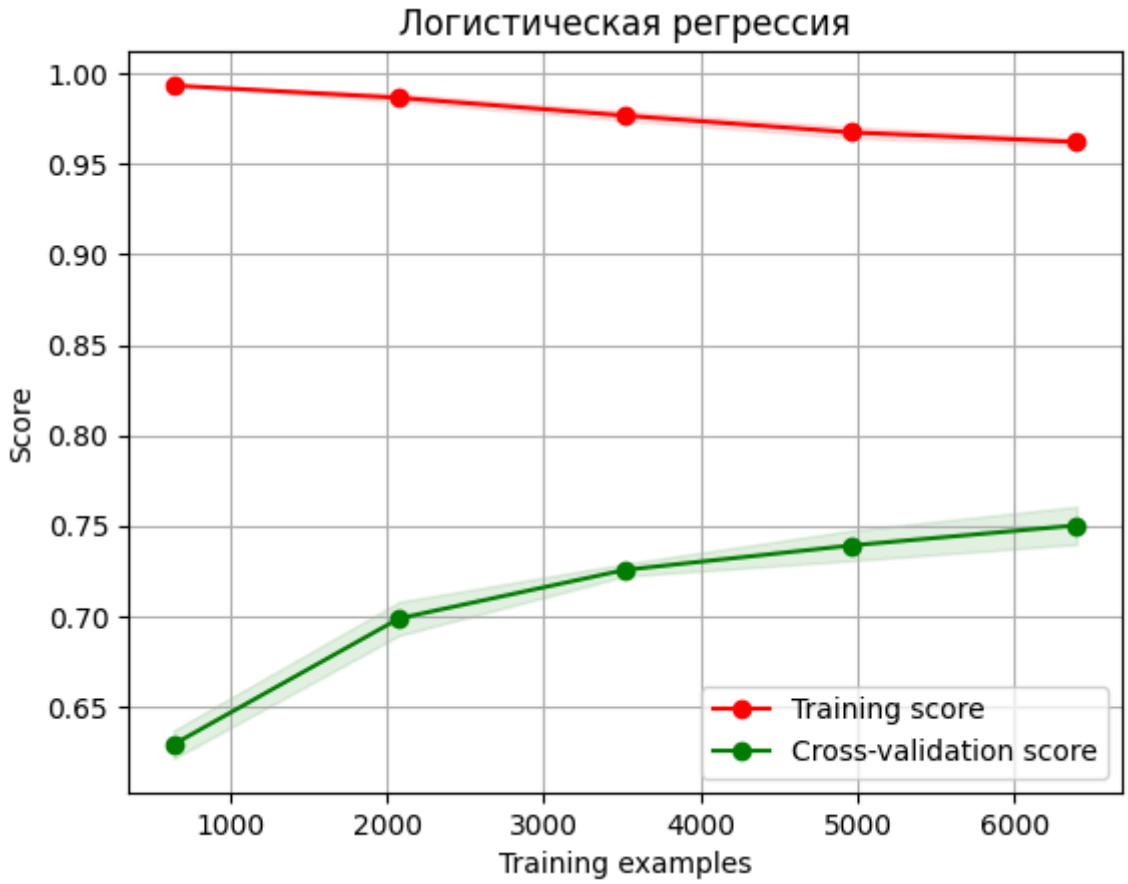
Время выполнения расчетов ниже: 10 минут

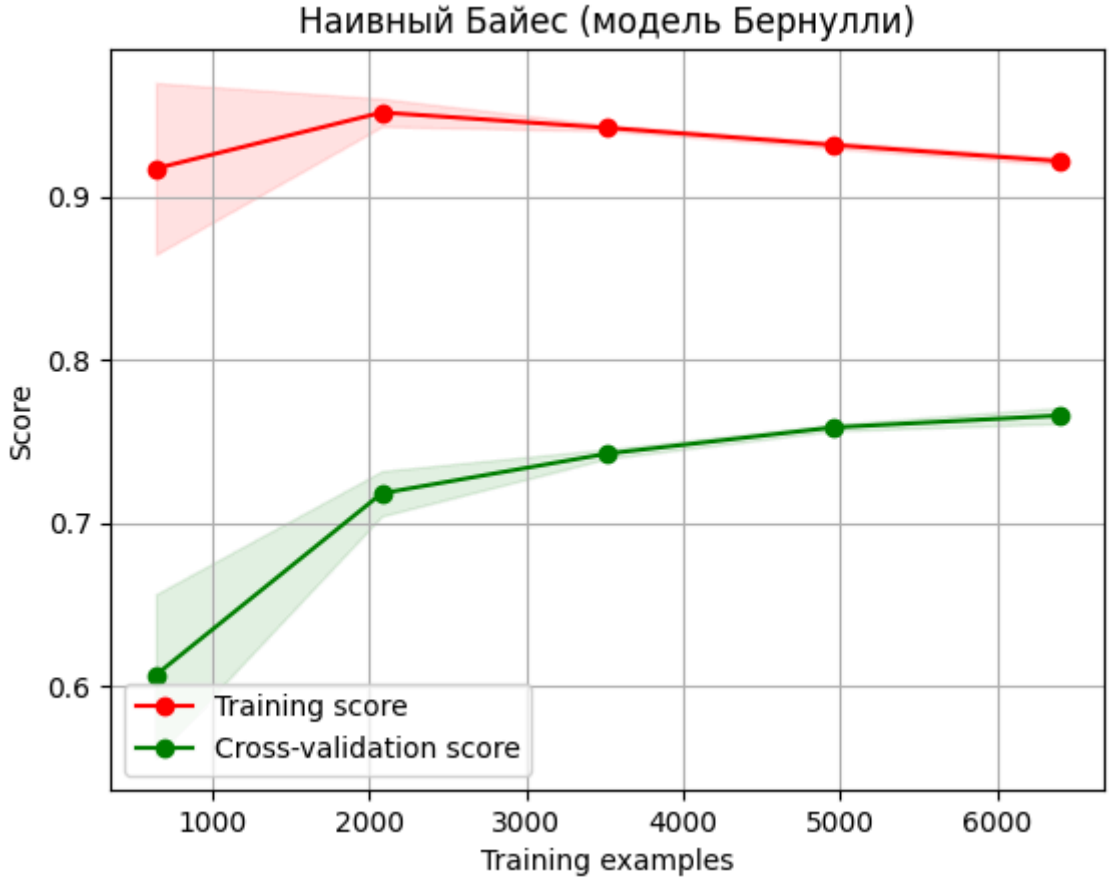
In [ ]:
```
# Создание пустого DataFrame для общих результатов
all_results_df = pd.DataFrame()
```

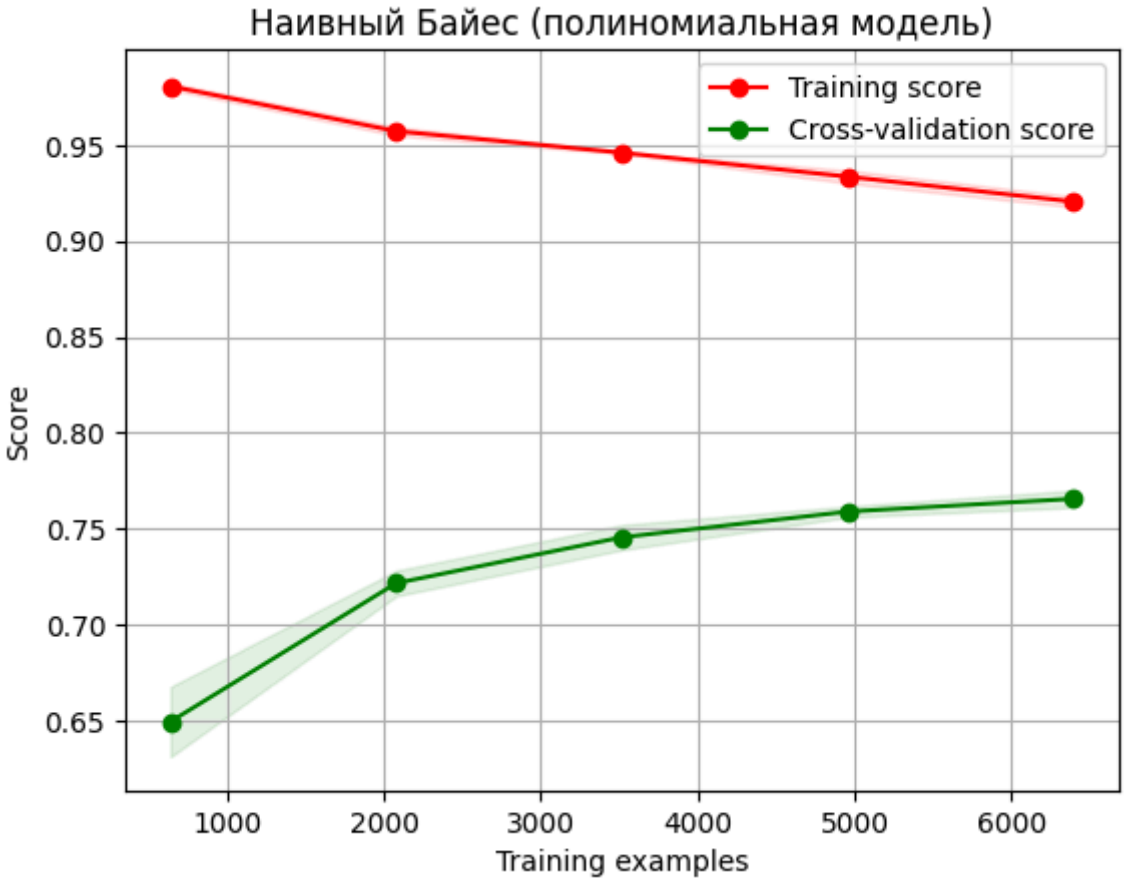
```python
# Запуск кода с разными значениями ngram
for ngram in [(1, 1), (2, 2), (1, 2)]:

    print(f"-----------------Параметр n-gram={ngram}-----------------")

    find_best_params_for_classificators(ngram)
    # Рисование графиков
    plot_all_curves(ngram)
    # Тренировка моделей
    results = train_models()

    # Преобразование результатов в DataFrame
    results_df = pd.DataFrame(results).T

    # Добавление столбца с параметром ngram
    results_df['ngram'] = str(ngram)

    # Добавление результатов в общий DataFrame
    all_results_df = pd.concat([all_results_df, results_df])

# Вывод общей таблицы в Google Colab
all_results_df
```

```
-----------------Параметр n-gram=(1, 1)-----------------
Лучшие параметры для К-ближайших соседей: {'classifier__n_neighbors': 81}
Лучшие параметры для Логистической регрессии: {'classifier__C': 0.517947467923121}
Лучшие параметры для Наивного Байеса (модель Бернулли): {'classifier__alpha': 1.93
06977288832496}
Лучшие параметры для Наивного Байеса (полиномиальная модель): {'classifier__alph
a': 1.9306977288832496}
```



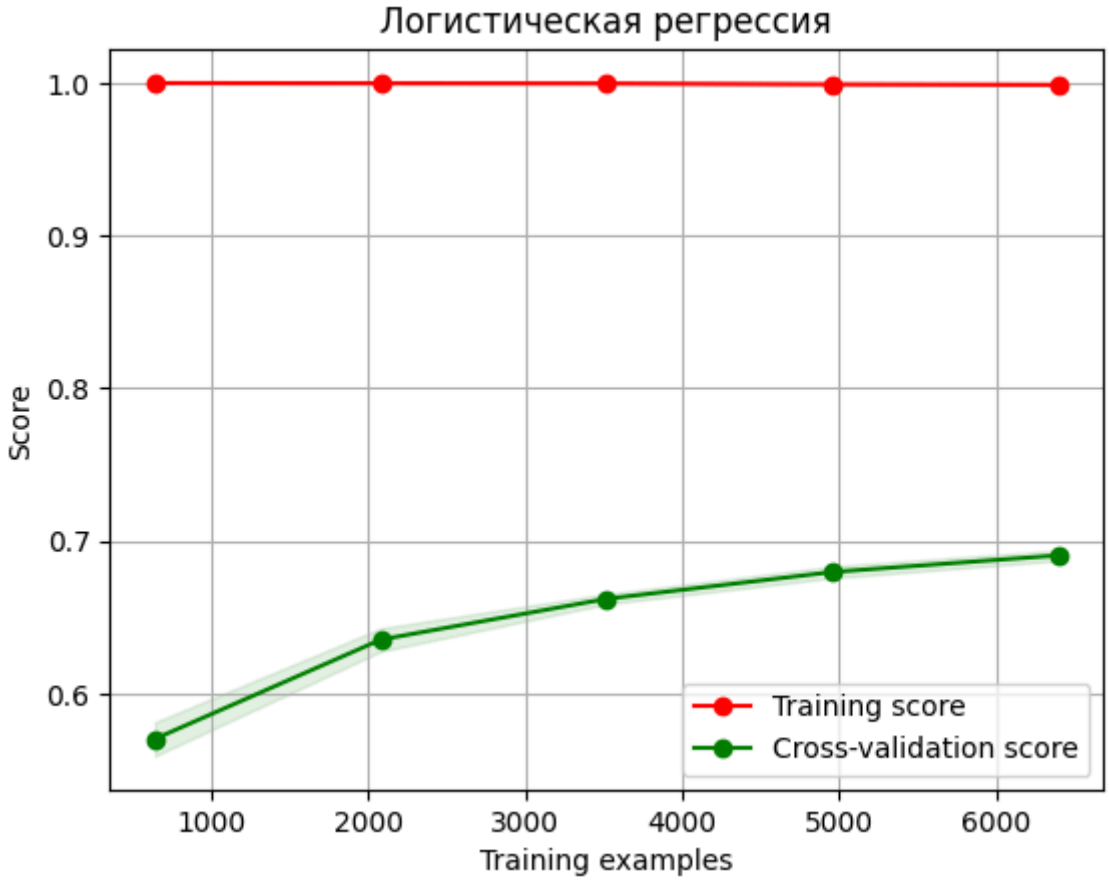

Логистическая регрессия
1.00
0.95
0.90
0.85
0.80
0.75
0.70
0.65
Score
1000
2000
3000
4000
5000
6000
Training examples
Training score
Cross-validation score

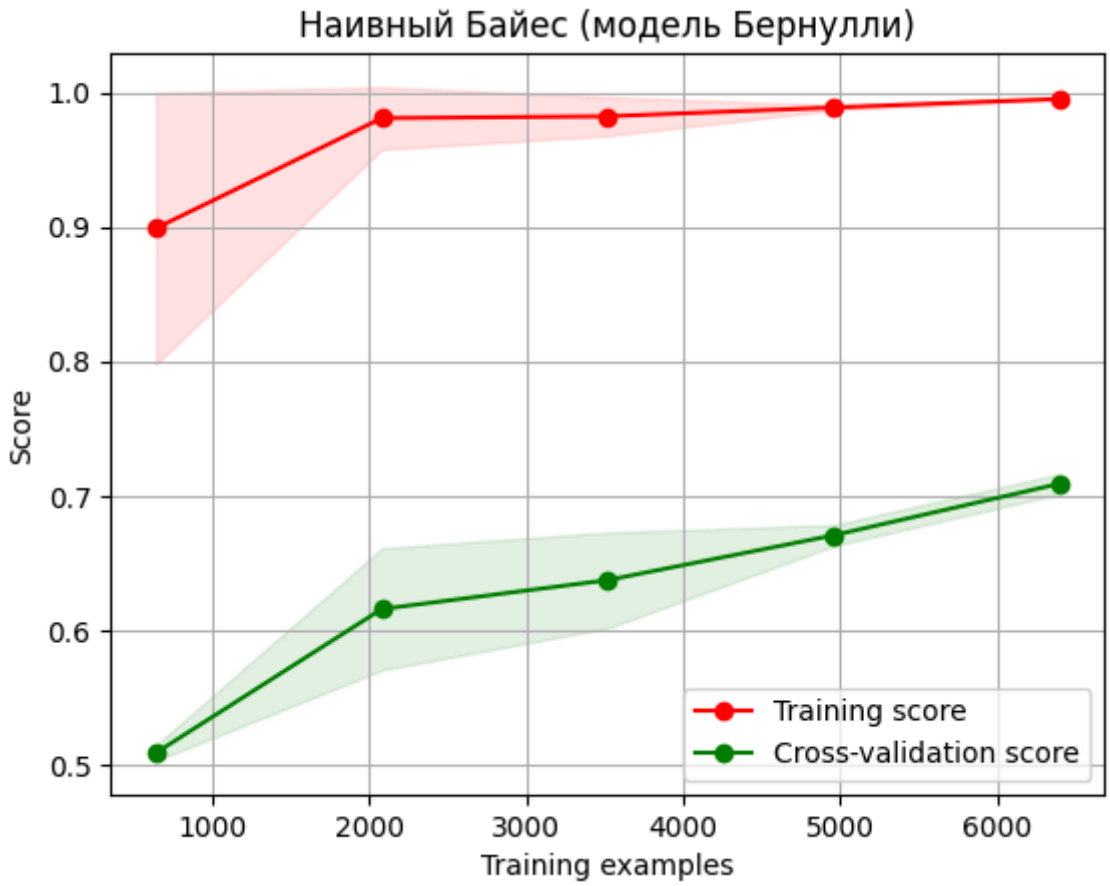

Наивный Байес (модель Бернулли)
0.9
0.8
0.7
0.6
Score
1000
2000
3000
4000
5000
6000
Training examples
Training score
Cross-validation score

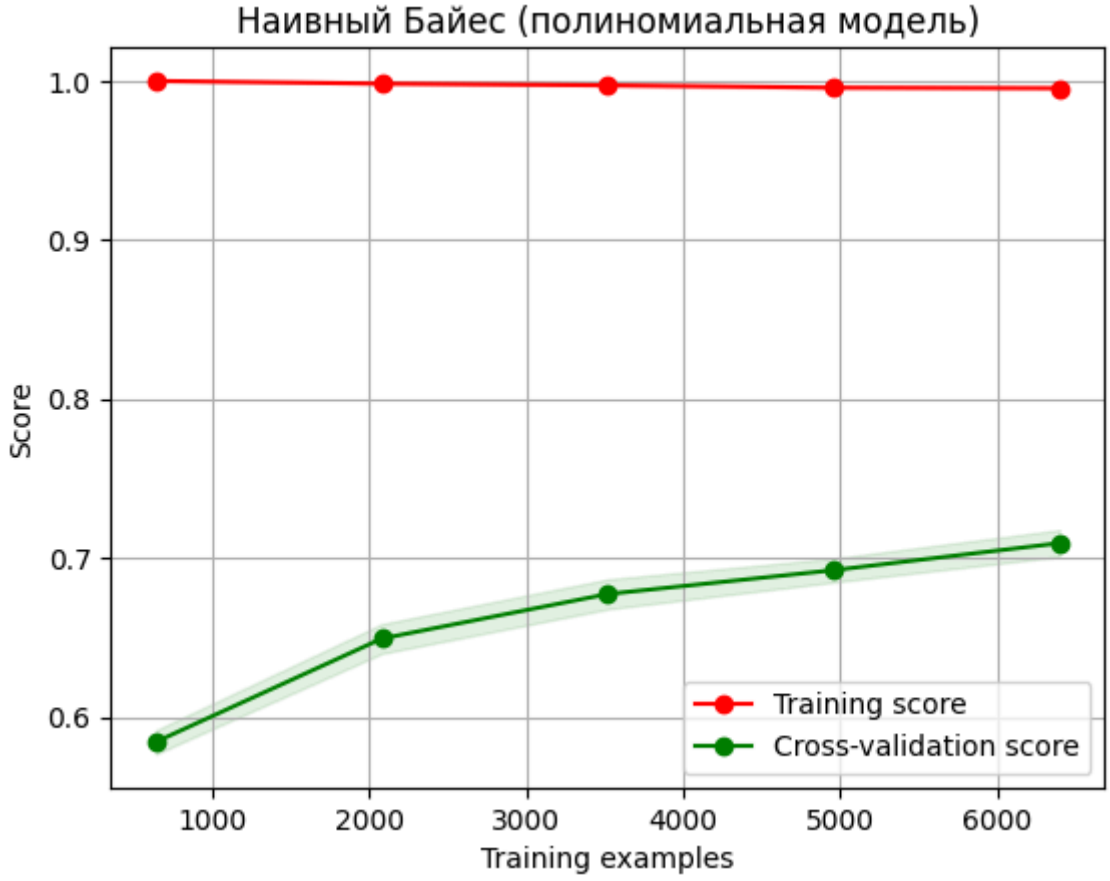

Наивный Байес (полиномиальная модель)
Training score
Cross-validation score
Score
0.95
0.90
0.85
0.80
0.75
0.70
0.65
1000
2000
3000
4000
5000
6000
Training examples

```
Модель: К-ближайших соседей
Accuracy: 0.5785
Precision: 0.5734
Recall: 0.6116
F1: 0.5919
Training Time: 0.2119 сек
Prediction Time: 0.9951 сек

Модель: Логистическая регрессия
Accuracy: 0.7754
Precision: 0.7771
Recall: 0.7720
F1: 0.7746
Training Time: 0.9994 сек
Prediction Time: 0.1243 сек

Модель: Наивный Байес (модель Бернулли)
Accuracy: 0.7740
Precision: 0.7903
Recall: 0.7458
F1: 0.7674
Training Time: 0.1970 сек
Prediction Time: 0.0412 сек

Модель: Наивный Байес (полиномиальная модель)
Accuracy: 0.7782
Precision: 0.7881
Recall: 0.7608
F1: 0.7742
Training Time: 0.2049 сек
Prediction Time: 0.0396 сек

-----------------Параметр n-gram=(2, 2)-----------------
Лучшие параметры для К-ближайших соседей: {'classifier__n_neighbors': 1}
Лучшие параметры для Логистической регрессии: {'classifier__C': 26.82695795279724
7}
Лучшие параметры для Наивного Байеса (модель Бернулли): {'classifier__alpha': 1.93
06977288832496}
Лучшие параметры для Наивного Байеса (полиномиальная модель): {'classifier__alph
a': 1.9306977288832496}
```

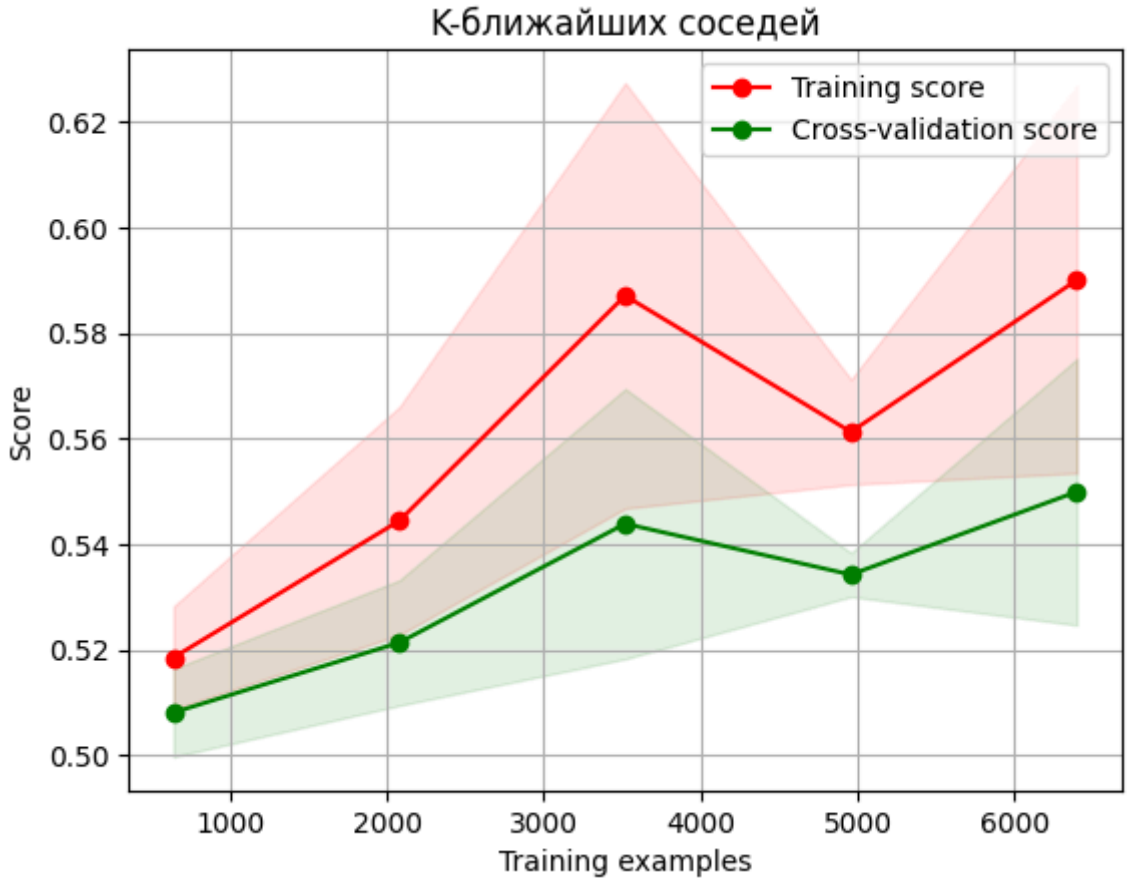

K-ближайших соседей
Training score
Cross-validation score
Score
0.70
0.65
0.60
0.55
0.50
1000
2000
3000
4000
5000
6000
Training examples

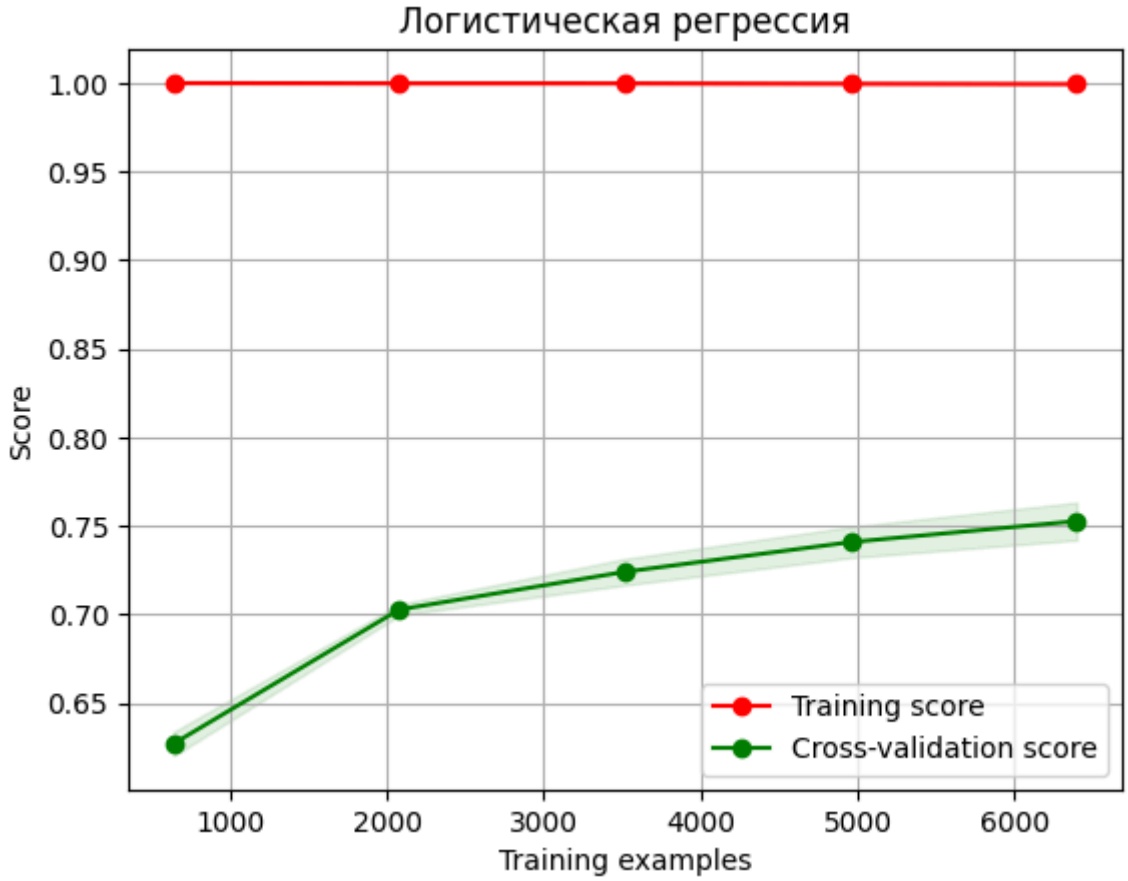

Логистическая регрессия
Score
1.0
0.9
0.8
0.7
0.6
Training score
Cross-validation score
1000
2000
3000
4000
5000
6000
Training examples

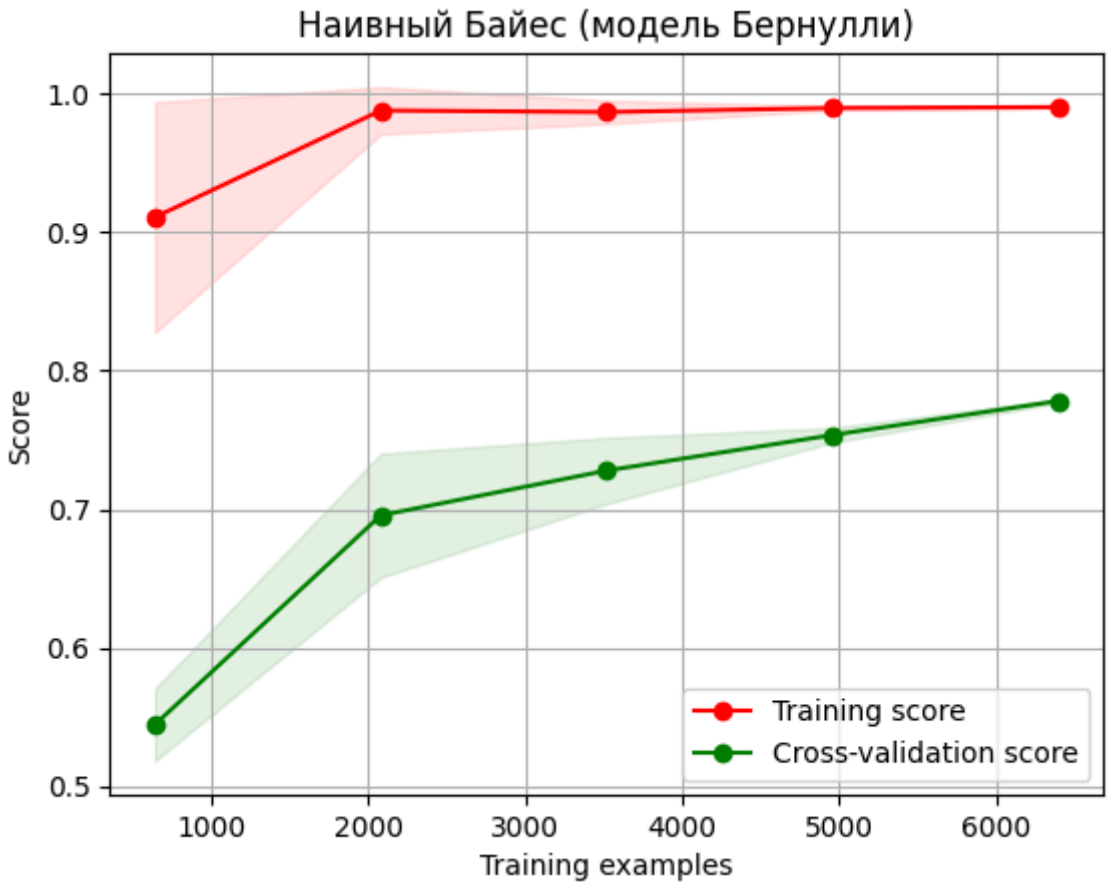

Наивный Байес (модель Бернулли)
Score
Training examples
Training score
Cross-validation score
1.0
0.9
0.8
0.7
0.6
0.5
1000
2000
3000
4000
5000
6000


Наивный Байес (полиномиальная модель)
Score
Training examples
Training score
Cross-validation score
1.0
0.9
0.8
0.7
0.6
1000
2000
3000
4000
5000
6000

```
Модель: К-ближайших соседей
Accuracy: 0.5785
Precision: 0.5734
Recall: 0.6116
F1: 0.5919
Training Time: 0.2493 сек
Prediction Time: 1.2669 сек

Модель: Логистическая регрессия
Accuracy: 0.7754
Precision: 0.7771
Recall: 0.7720
F1: 0.7746
Training Time: 1.8709 сек
Prediction Time: 0.1531 сек

Модель: Наивный Байес (модель Бернулли)
Accuracy: 0.7740
Precision: 0.7903
Recall: 0.7458
F1: 0.7674
Training Time: 0.3439 сек
Prediction Time: 0.0723 сек

Модель: Наивный Байес (полиномиальная модель)
Accuracy: 0.7782
Precision: 0.7881
Recall: 0.7608
F1: 0.7742
Training Time: 0.3559 сек
Prediction Time: 0.0710 сек

-----------------Параметр n-gram=(1, 2)-----------------
Лучшие параметры для К-ближайших соседей: {'classifier__n_neighbors': 41}
Лучшие параметры для Логистической регрессии: {'classifier__C': 1389.495494373136
1}
Лучшие параметры для Наивного Байеса (модель Бернулли): {'classifier__alpha': 1.93
06977288832496}
Лучшие параметры для Наивного Байеса (полиномиальная модель): {'classifier__alph
a': 0.3727593720314942}
```


K-ближайших соседей
Training score
Cross-validation score
Score
0.62
0.60
0.58
0.56
0.54
0.52
0.50
1000
2000
3000
4000
5000
6000
Training examples


Логистическая регрессия
Score
1.00
0.95
0.90
0.85
0.80
0.75
0.70
0.65
Training score
Cross-validation score
1000
2000
3000
4000
5000
6000
Training examples


Наивный Байес (модель Бернулли)
Score
Training examples
Training score
Cross-validation score
1.0
0.9
0.8
0.7
0.6
0.5
1000
2000
3000
4000
5000
6000


Наивный Байес (полиномиальная модель)
Score
Training examples
Training score
Cross-validation score
1.00
0.95
0.90
0.85
0.80
0.75
0.70
0.65
1000
2000
3000
4000
5000
6000

Модель: К-ближайших соседей
Accuracy: 0.5785
Precision: 0.5734
Recall: 0.6116
F1: 0.5919
Training Time: 0.2003 сек
Prediction Time: 0.9781 сек

Модель: Логистическая регрессия
Accuracy: 0.7754
Precision: 0.7771
Recall: 0.7720
F1: 0.7746
Training Time: 0.9525 сек
Prediction Time: 0.1261 сек

Модель: Наивный Байес (модель Бернулли)
Accuracy: 0.7740
Precision: 0.7903
Recall: 0.7458
F1: 0.7674
Training Time: 0.1899 сек
Prediction Time: 0.0410 сек

Модель: Наивный Байес (полиномиальная модель)
Accuracy: 0.7782
Precision: 0.7881
Recall: 0.7608
F1: 0.7742
Training Time: 0.1967 сек
Prediction Time: 0.0398 сек

Out[ ]:

	Accuracy	Precision	Recall	F1	Training Time	Prediction Time	ngram
K-ближайших соседей	0.578528	0.573439	0.611632	0.591920	0.211898	0.995055	(1, 1)
Логистическая регрессия	0.775434	0.777148	0.772045	0.774588	0.999403	0.124328	(1, 1)
Наивный Байес (модель Бернулли)	0.774027	0.790258	0.745779	0.767375	0.196989	0.041212	(1, 1)
Наивный Байес (полиномиальная модель)	0.778247	0.788144	0.760788	0.774224	0.204852	0.039622	(1, 1)
K-ближайших соседей	0.578528	0.573439	0.611632	0.591920	0.249281	1.266949	(2, 2)
Логистическая регрессия	0.775434	0.777148	0.772045	0.774588	1.870860	0.153117	(2, 2)
Наивный Байес (модель Бернулли)	0.774027	0.790258	0.745779	0.767375	0.343920	0.072349	(2, 2)
Наивный Байес (полиномиальная модель)	0.778247	0.788144	0.760788	0.774224	0.355918	0.070976	(2, 2)
K-ближайших соседей	0.578528	0.573439	0.611632	0.591920	0.200304	0.978097	(1, 2)
Логистическая регрессия	0.775434	0.777148	0.772045	0.774588	0.952512	0.126081	(1, 2)
Наивный Байес (модель Бернулли)	0.774027	0.790258	0.745779	0.767375	0.189855	0.041049	(1, 2)
Наивный Байес (полиномиальная модель)	0.778247	0.788144	0.760788	0.774224	0.196651	0.039840	(1, 2)

На основании представленных результатов можно сделать несколько выводов:

1. Значения параметров моделей (например, `n_neighbors` для K-ближайших соседей) меняются в зависимости от параметра `n-gram`, что может быть связано с тем, как различные наборы параметров влияют на качество модели для конкретного типа данных.
2. Время обучения и предсказания моделей, различается для разных наборов параметров `n-gram`. Например, время обучения и предсказания для моделей с `n-gram=(1, 1)` и `n-gram=(1, 2)` в целом меньше, чем для моделей с `n-gram=(2, 2)`. Это может быть связано с тем, что больший размер n-gram увеличивает количество признаков и, следовательно, время вычислений.
3. Все модели демонстрируют схожие результаты. Однако например, наивный Байес имеет более низкого времени обработки, а логистическая регрессия предсказывает точнее.