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 **КОМПЬЮТЕРНЫЕ СИСТЕМЫ И СЕТИ (ИУ6)**

НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.04.01 Информатика и вычислительная техника**

МАГИСТЕРСКАЯ ПРОГРАММА **09.04.01/07 Интеллектуальные системы анализа, обработки и интерпретации больших данных**

О Т Ч Е Т

по домашнему заданию № 2

Название: Модели предсказания

Дисциплина: Методы машинного обучения

Студент ИУ6-22М _____ Т.И. Кадыров
(Группа) (Подпись, дата) (И.О. Фамилия)

Преподаватель _____ С.Ю. Папулин
(Подпись, дата) (И.О. Фамилия)

Москва, 2024

ДОМАШНЕЕ ЗАДАНИЕ 2. Модели предсказания

Кадыров Т.И. ИУ6-22М

Цель работы

Приобрести опыт решения практических задач по машинному обучению, таких как анализ и визуализация исходных данных, обучение, выбор и оценка качества моделей предсказания, посредством языка программирования Python.

При выполнении работы решаются следующие задачи:

- реализация собственных классов совместимых с библиотекой `sklearn`
- оценка влияния регуляризации в моделях предсказания
- преобразование исходных данных посредством транформаторов `sklearn`
- использование отложенной выборки и кросс-валидации
- выбор гиперпараметров и интерпретация кривых обучения
- оценка качества моделей предсказания
- выявление преимуществ и недостатков методов предсказания в зависимости от поставленной задачи

Вариант

Чтобы узнать свой вариант, введите Вашу фамилию в соответствующее поле ниже и запустите ячейку:

In [ ]:
```
surname = "Кадыров"  # 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:  2
```

Задача 1. Реализация собственных классов и функций

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 [ ]:
```
import numpy as np
```

In [ ]:
```
# Класс модели линейной регрессии с регуляризацией Тихонова
class LinearRegression:
    def __init__(self, alpha):
        # Коэффициент регуляризации
        self.alpha = alpha
        # Коэффициенты модели
        self.weights = None
    def fit(self, X, y):
        # Матрица регуляризации
        A = np.eye(X.shape[1])
        # Не регуляризируем свободный коэффициент
        A[0, 0] = 0
        # Решаем систему линейных уравнений регуляризованного метода наименьших квадратов
        self.weights = np.linalg.inv(X.T @ X + self.alpha * A) @ X.T @ y
    def predict(self, X):
        # Предсказание на основе весов модели
        return np.dot(X, self.weights)
```

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 [ ]:
```
import warnings
warnings.filterwarnings('ignore')
```

In [ ]:
```
# Класс для стандартизации признаков по методу Z-score
class FeatureStandardScaler:
    def __init__(self, has_bias, apply_mean):
        self.has_bias_ = has_bias
        self.apply_mean_ = apply_mean
        # Среднее арифметическое
        self.mean_ = None
        # Стандартное отклонение
        self.std_ = None
    def fit(self, X, y=None):
        # Высчитываем среднее по каждому признаку
        self.mean_ = np.mean(X, axis=0)
        # Считаем стандартное отклонение
        self.std_ = np.std(X, axis=0)
        return self
    def transform(self, X):
        X_res = X.copy()
        # Стандартизируем
        if self.apply_mean_:
            X_res -= self.mean_
        X_res = X_res / self.std_
        # Добавляем единичный столбец если отсутствует
        if not self.has_bias_:
            X_res = np.insert(X_res, 0, 1, axis=1)
        return X_res
    def fit_transform(self, X, y=None):
        # Обучаем и трансформируем выборку на обученной модели
        return self.fit(X).transform(X)
```

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.model_selection import KFold
```

In [ ]:
```
# Функция для рассчета среднеквадратичной ошибки
def mse(y, y_pred):
    return ((y - y_pred) ** 2).mean()
# Функция для рассчета коэффициента детерминации
def r2_score(y, y_pred):
    ss_res = ((y - y_pred) ** 2).sum()
    ss_tot = ((y - y.mean()) ** 2).sum()
    return 1 - ss_res / ss_tot

# Функция для анализа оценки производительности модели при отложенной выборке
def run_holdout(model, X, y, train_size):
    idx = int(len(X)*train_size)
    # Разделяем выборку на обучающую и тестовую части
    X_train, X_test = X[:idx], X[idx:]
    y_train, y_test = y[:idx], y[idx:]
    # Обучаем модель
    model.fit(X_train, y_train)

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

    # Вычисляем MSE и R2 для обучающей и тестовой выборки
    mse_train = mse(y_train, y_train_pred)
    r2_train = r2_score(y_train, y_train_pred)
    mse_test = mse(y_test, y_test_pred)
    r2_test = r2_score(y_test, y_test_pred)

    return mse_train, r2_train, mse_test, r2_test

# Функция для анализа оценки производительности модели при кросс-валидации
def run_cross_val(model, X, y, n_splits, shuffle, random_state):
```

```
    cross_val = KFold(n_splits=n_splits, shuffle=shuffle, random_state=random_state)
    # Массивы оценок для каждой итерации обучения
    mse_train_list = list()
    r2_train_list = list()
    mse_test_list = list()
    r2_test_list = list()

    for idx_train, idx_test in cross_val.split(X):
        # Разделяем
        X_train, X_test = X[idx_train], X[idx_test]
        y_train, y_test = y[idx_train], y[idx_test]
        # Обучаем
        model.fit(X_train, y_train)
        # Предсказываем
        y_train_pred = model.predict(X_train)
        y_test_pred = model.predict(X_test)
        # Считаем оценки
        mse_train_list.append(mse(y_train, y_train_pred))
        r2_train_list.append(r2_score(y_train, y_train_pred))
        mse_test_list.append(mse(y_test, y_test_pred))
        r2_test_list.append(r2_score(y_test, y_test_pred))

    # Считаем средние оценки
    mse_train_avg = np.mean(mse_train_list)
    r2_train_avg = np.mean(r2_train_list)
    mse_test_avg = np.mean(mse_test_list)
    r2_test_avg = np.mean(r2_test_list)

    return mse_train_avg, r2_train_avg, mse_test_avg, r2_test_avg
```

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 ($\hat{y}$) - 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
import matplotlib.pyplot as plt
import pandas as pd
```

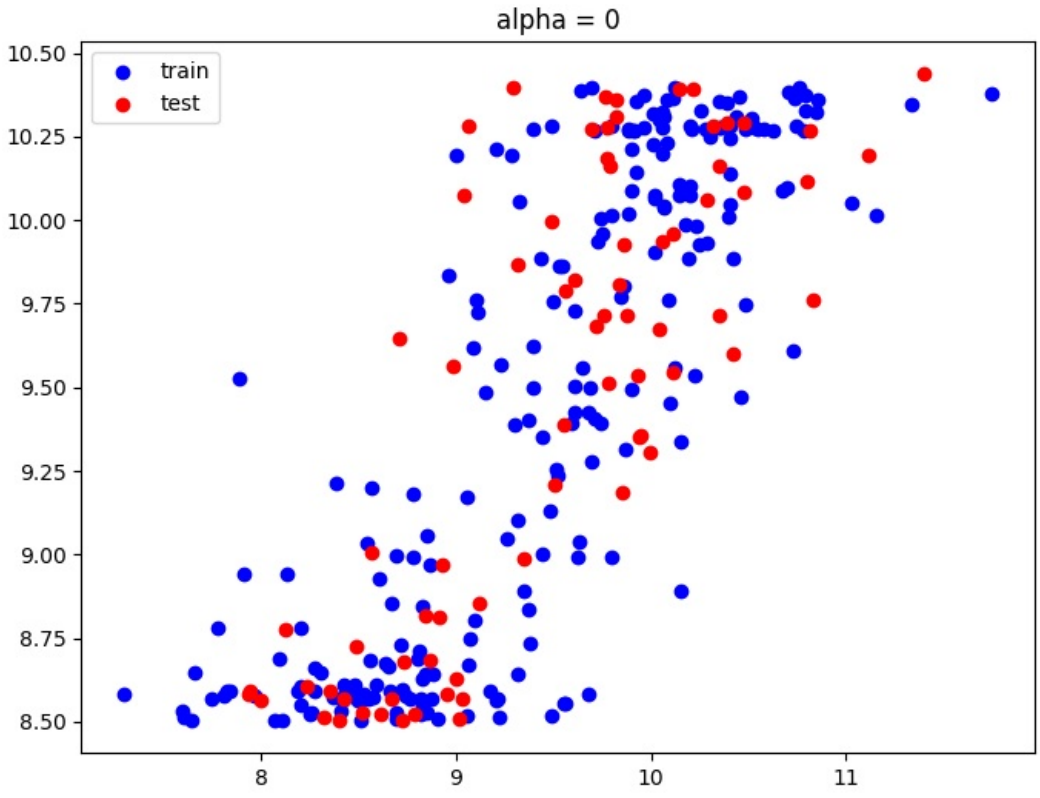
In [ ]:
```
# Загружаем исходные данные
df = pd.read_csv("data/regularization.csv")

# Выделяем признаки и целевое значение
X = df[df.filter(regex='^X').columns].to_numpy()
y = df["Y"].to_numpy()

# has_bias = False, так как необходимо добавить единичный столбец для свободного коэффициента
model_1_ho = Pipeline([
    ('scaler', FeatureStandardScaler(False, True)),
    ('regression', LinearRegression(0))
])

model_1_cv = Pipeline([
    ('scaler', FeatureStandardScaler(False, True)),
    ('regression', LinearRegression(0))
])

model_2_ho = Pipeline([
    ('scaler', FeatureStandardScaler(False, True)),
    ('regression', LinearRegression(0.01))
])

model_2_cv = Pipeline([
    ('scaler', FeatureStandardScaler(False, True)),
    ('regression', LinearRegression(0.01))
])


# Обучаем и оцениваем модели
mse_train_ho_1, r2_train_ho_1, mse_test_ho_1, r2_test_ho_1 = run_holdout(model_1_ho, X, y, 0.75)
mse_train_cv_1, r2_train_cv_1, mse_test_cv_1, r2_test_cv_1 = run_cross_val(model_1_cv, X, y, 4, True, 0)
mse_train_ho_2, r2_train_ho_2, mse_test_ho_2, r2_test_ho_2 = run_holdout(model_2_ho, X, y, 0.75)
mse_train_cv_2, r2_train_cv_2, mse_test_cv_2, r2_test_cv_2 = run_cross_val(model_2_cv, X, y, 4, True, 0)

# Выводим оценки
print("For model 1 (alpha=0):")
print(f"MSE (holdout): train={mse_train_ho_1:.4f} test={mse_test_ho_1:.4f}")
```

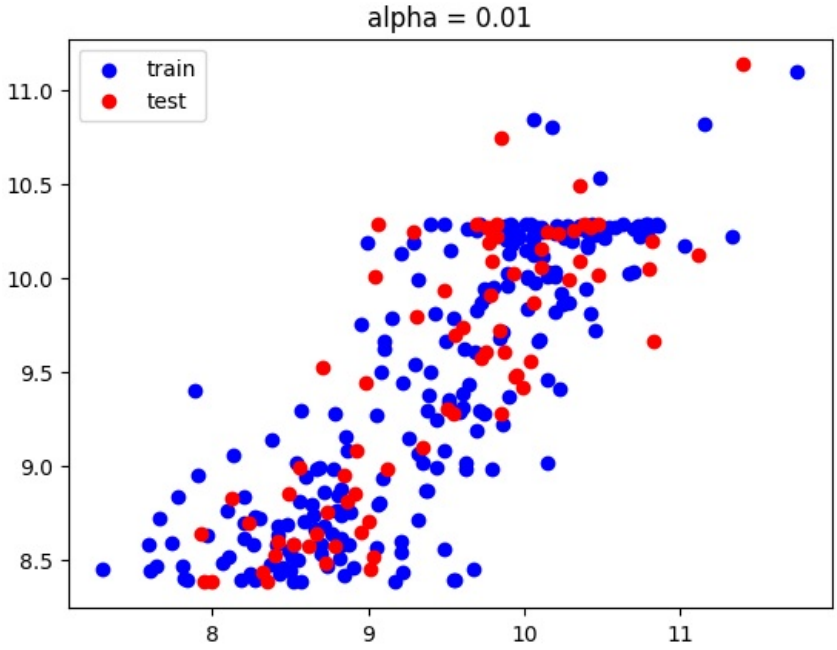
```
print(f"R2 (holdout): train={r2_train_ho_1:.4f} test={r2_test_ho_1:.4f}")
print(f"MSE (cross validation): train={mse_train_cv_1:.4f} test={mse_test_cv_1:.4f}")
print(f"R2 (cross validation): train={r2_train_cv_1:.4f} test={r2_test_cv_1:.4f}")
print("==================================================")
print("For model 2 (alpha=0.01):")
print(f"MSE (holdout): train={mse_train_ho_2:.4f} test={mse_test_ho_2:.4f}")
print(f"R2 (holdout): train={r2_train_ho_2:.4f} test={r2_test_ho_2:.4f}")
print(f"MSE (cross validation): train={mse_train_cv_2:.4f} test={mse_test_cv_2:.4f}")
print(f"R2 (cross validation): train={r2_train_cv_2:.4f} test={r2_test_cv_2:.4f}")
```

```
For model 1 (alpha=0):
MSE (holdout): train=0.2499 test=0.2442
R2 (holdout): train=0.6667 test=0.6175
MSE (cross validation): train=0.2527 test=0.2733
R2 (cross validation): train=0.6498 test=0.6192
==================================================
For model 2 (alpha=0.01):
MSE (holdout): train=0.2340 test=0.2045
R2 (holdout): train=0.6878 test=0.6796
MSE (cross validation): train=0.2235 test=0.2365
R2 (cross validation): train=0.6903 test=0.6706
```

In [ ]:

```
# Разделяем выборки на test и train, train_size=0.75
idx = int(0.75*len(X))
y_train, y_test = y[:idx], y[idx:]
X_train, X_test = X[:idx], X[idx:]
# Используем модель, обученную методом holdout, так как там такое же разделение
y_train_pred_1 = model_1_ho.predict(X_train)
y_test_pred_1 = model_1_ho.predict(X_test)
y_train_pred_2 = model_2_ho.predict(X_train)
y_test_pred_2 = model_2_ho.predict(X_test)
# Выводим графики зависимости предсказанных
plt.figure(figsize=(8, 6))
plt.scatter(y_train, y_train_pred_1, label='train', color='blue')
plt.scatter(y_test, y_test_pred_1, label='test', color='red')
plt.title('alpha = 0')
plt.legend()
plt.show()
plt.scatter(y_train, y_train_pred_2, label='train', color='blue')
plt.scatter(y_test, y_test_pred_2, label='test', color='red')
plt.title('alpha = 0.01')
plt.legend()
plt.show()
```





Задача 2. Классификация и кросс-валидация

Вариант 2

⚠ **Замечание**:

- Используйте класс логистической регрессии из `sklearn` со следующими параметрами:
  - `penalty='l2'`
  - `fit_intercept=True`
  - `max_iter=100`
  - `C=1e5`
  - `solver='liblinear'`
  - `random_state=12345`
- Разбейте исходные данные на обучающее и тестовое подмножества в соотношении `70` на `30`, `random_state=0`
- Для выбора гиперпараметров используйте два подхода: 1) с отложенной выборкой, 2) с кросс-валидацией
- Для кросс-валидации использовать функцию `cross_validate` из `sklearn`
- Параметры разбиения для выбора гиперпараметров используйте те, что в п.4 задачи 1

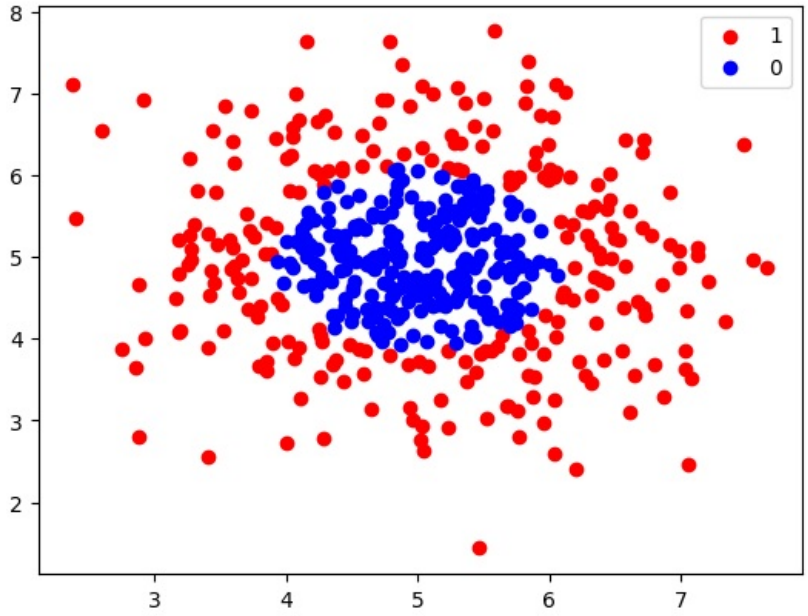
Дано множество наблюдений (см. набор данных к заданию), классификатор - логистическая регрессия. Найти степень полинома с минимальной ошибкой на проверочном подмножестве. Для лучшего случая рассчитать ошибку на тестовом подмножестве. В качестве метрики использовать долю правильных классификаций. Сделать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

Построить:

- диаграмму разброса исходных данных
- зависимость доли правильных классификаций от степени полинома для обучающего и проверочного подмножеств (две кривые на одном графике)
- результат классификации для наилучшего случая (степень полинома) для обучающего и тестового подмножеств с указанием границы принятия решения

In [ ]:
```
# Загружаем исходные данные
df = pd.read_csv("data/Cl_A5_V2.csv")
X = df[df.filter(regex='^X').columns].to_numpy()
y = df['Y'].to_numpy()

# Строим диаграмму разброса исходных данных
plt.scatter(df[df['Y'] == 1.0]['X1'].to_numpy(), df[df['Y'] == 1.0]['X2'].to_numpy(), label='1', color='red')
plt.scatter(df[df['Y'] == 0.0]['X1'].to_numpy(), df[df['Y'] == 0.0]['X2'].to_numpy(), label='0', color='blue')
plt.legend()
```

```
plt.show()
```



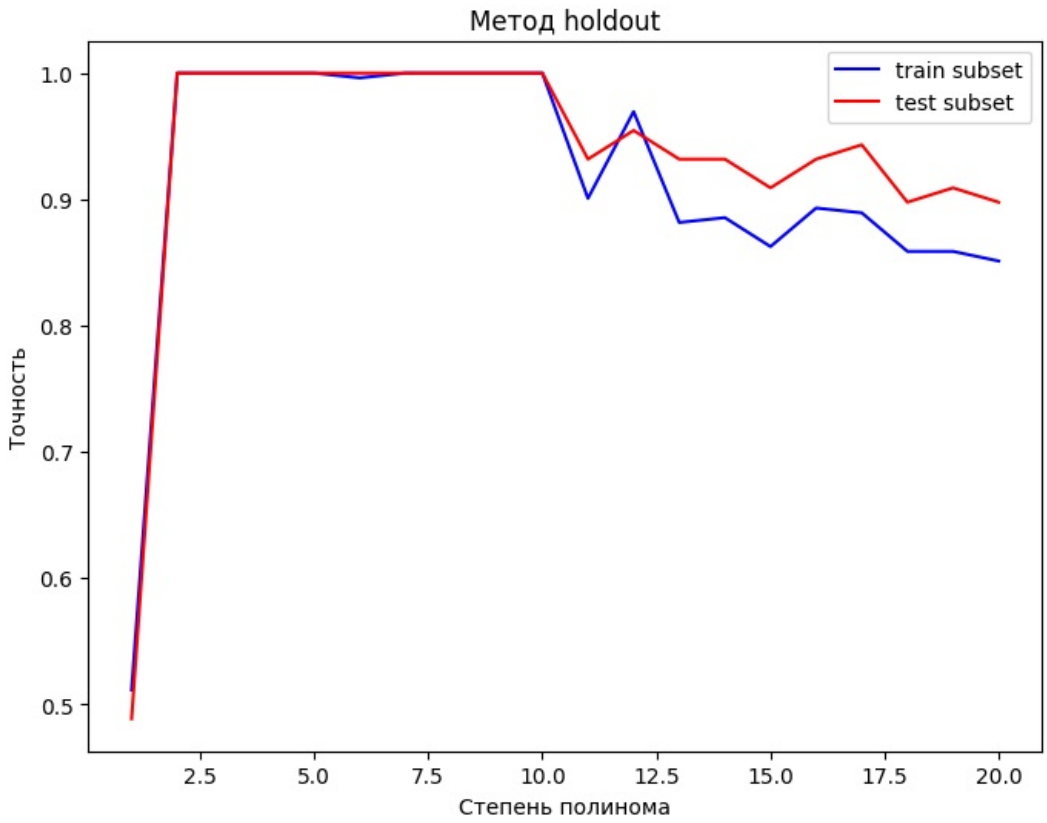
In [ ]:
```
from sklearn.model_selection import train_test_split, cross_validate
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import accuracy_score
```

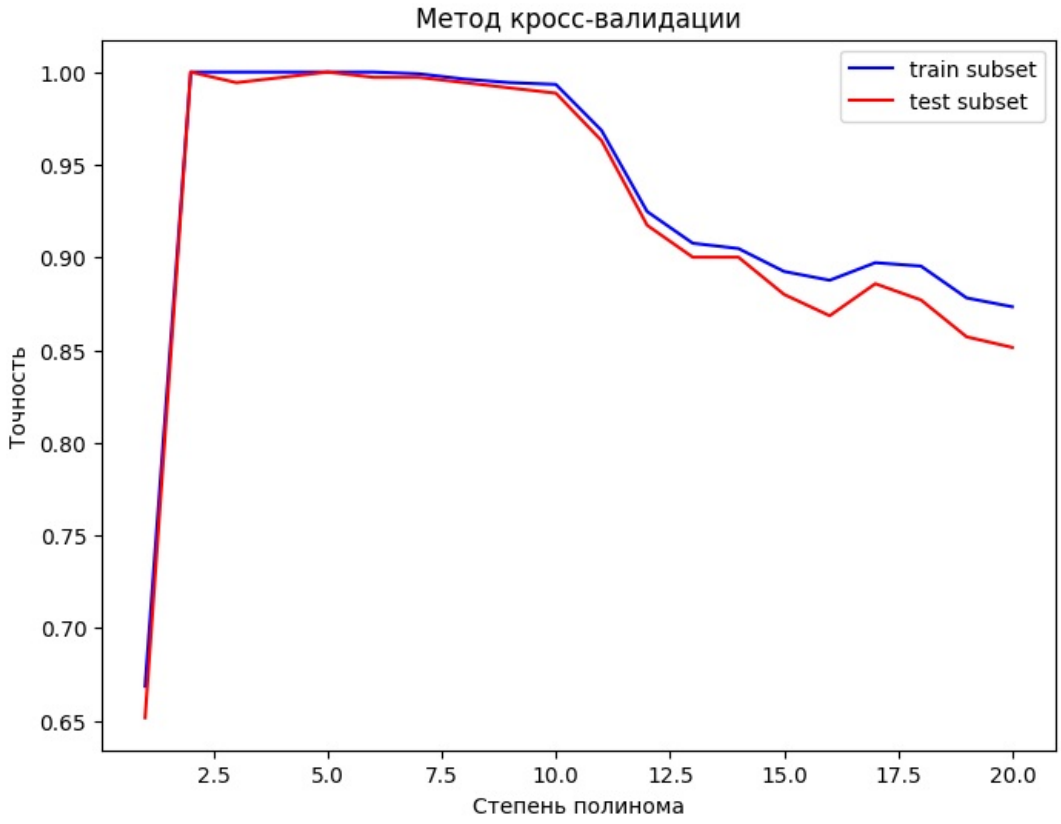
In [ ]:
```
# Разбиваем данные на обучающую/валидационную и тестовую выборки
X_train_val, X_test, y_train_val, y_test = train_test_split(X, y, train_size=0.7, random_state=0)
# Разделяем выборку на обучающую и валидационную для метода отложенной выборки
X_train, X_val, y_train, y_val = train_test_split(X_train_val, y_train_val, train_size=0.75, random_state=0)
# Лучшая степень полинома для двух разных методов
best_degree_ho = None
best_accur_ho = -1
best_degree_cv = None
best_accur_cv = -1
# Список оценок для разных методов
train_accur_ho_list = list()
test_accur_ho_list = list()
train_accur_cv_list = list()
test_accur_cv_list = list()
# Переберем 20 степеней полинома (чтобы показать, где точность будет падать)
degrees = list(range(1, 21))
for degree in degrees:
    poly = PolynomialFeatures(degree=degree)
    # 1) Выбор гиперпараметров путем отложенной выборки
    X_train_poly = poly.fit_transform(X_train)
    X_val_poly = poly.transform(X_val)
    # Создаем модель
    model_ho = LogisticRegression(penalty='l2', fit_intercept=True, max_iter=100, C=1e5, solver='liblinear', ra
    # Обучаем
    model_ho.fit(X_train_poly, y_train)
    # Тестируем
    y_val_pred = model_ho.predict(X_val_poly)
    # Оцениваем
    test_accur_ho = accuracy_score(y_val, y_val_pred)
    # Если лучше, то сохраняем результат
    if test_accur_ho > best_accur_ho:
        best_accur_ho = test_accur_ho
        best_degree_ho = degree
    # Сохраняем результат точности для графика зависимости от степени полинома
    y_train_pred = model_ho.predict(X_train_poly)
    train_accur_ho_list.append(accuracy_score(y_train, y_train_pred))
    test_accur_ho_list.append(test_accur_ho)
    # 2) Выбор гиперпараметров путем кросс-валидации
    X_train_val_poly = poly.fit_transform(X_train_val)
    # Создаем модель
    model_cv = LogisticRegression(penalty='l2', fit_intercept=True, max_iter=100, C=1e5, solver='liblinear', ra
    # Обучаем
    cv_results = cross_validate(model_cv, X_train_val_poly, y_train_val, cv=4, scoring='accuracy', return_train_
    # Оцениваем
    test_accur_cv = np.mean(cv_results['test_score'])
    # Сохраняем лучший полином
    if test_accur_cv > best_accur_cv:
        best_accur_cv = test_accur_cv
        best_degree_cv = degree
    # Сохраняем результаты
    train_accur_cv_list.append(np.mean(cv_results['train_score']))
```

```
    test_accur_cv_list.append(test_accur_cv)

# Лучшая степень полинома для двух разных методов
best_degree_ho, best_degree_cv
```

Out[ ]: (2, 2)

In [ ]:
```
# Строим график зависимости точности модели от степени полинома для отложенной выборки
plt.figure(figsize=(8, 6))
plt.xlabel("Степень полинома")
plt.ylabel("Точность")
plt.title("Метод holdout")
plt.plot(degrees, train_accur_ho_list, label='train subset', color='blue')
plt.plot(degrees, test_accur_ho_list, label='test subset', color='red')
plt.legend()
plt.show()

# Строим график зависимости точности модели от степени полинома для кросс-валидации
plt.figure(figsize=(8, 6))
plt.xlabel("Степень полинома")
plt.ylabel("Точность")
plt.title("Метод кросс-валидации")
plt.plot(degrees, train_accur_cv_list, label='train subset', color='blue')
plt.plot(degrees, test_accur_cv_list, label='test subset', color='red')
plt.legend()
plt.show()
```



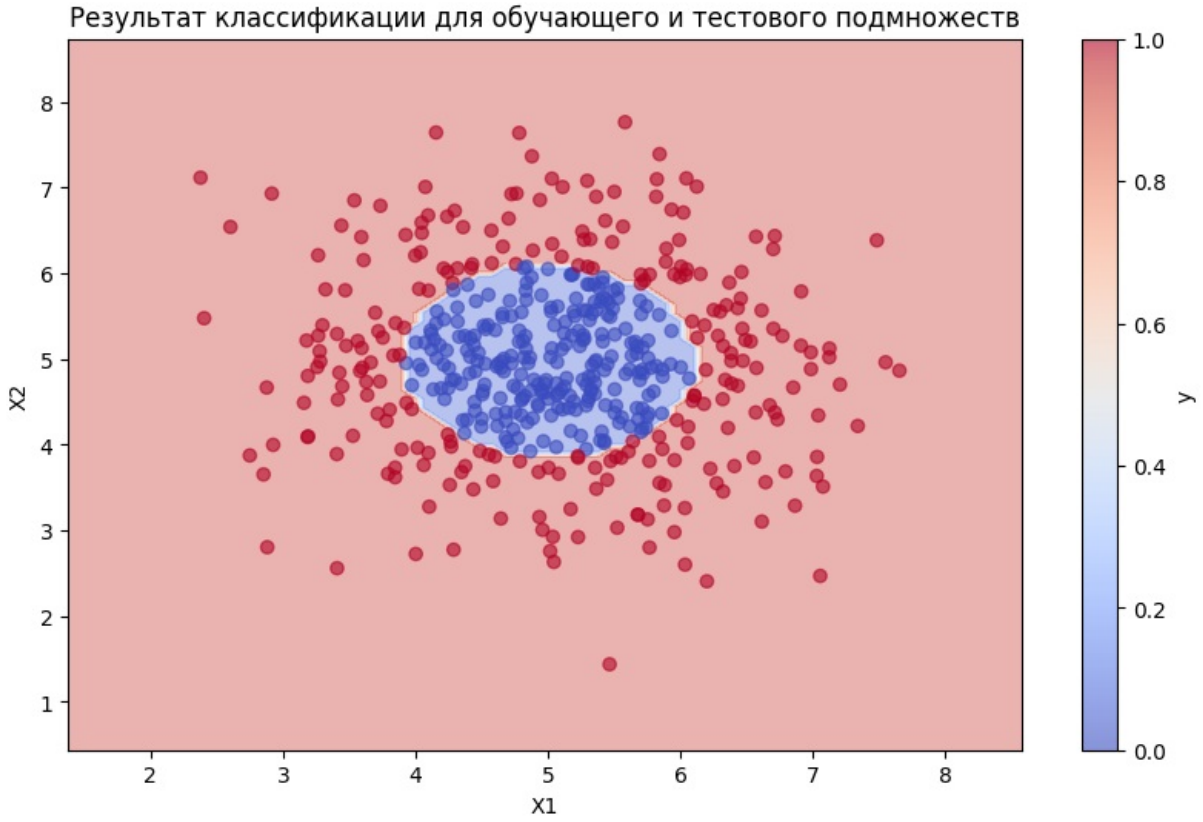


In [ ]:
```
# Проверяем итоговую модель на тестовой выборке
model = LogisticRegression(penalty='l2', fit_intercept=True, max_iter=100, C=1e5, solver='liblinear', random_sta
poly = PolynomialFeatures(degree=best_degree_ho)
X_train_poly = poly.fit_transform(X_train)
model.fit(X_train_poly, y_train)
X_test_poly = poly.transform(X_test)
# Выводим оценку эффективности модели на тестовом подмножестве
accur_test_score = model.score(X_test_poly, y_test)
accur_test_score
```

Out[ ]: 1.0

In [ ]:
```
# Предсказываем для все выборки
X_poly = poly.transform(X)
y_pred = model.predict(X_poly)

# Определение минимального и максимального значений признаков
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1

# Генерация сетки точек для построения контурного графика
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.1),
                     np.arange(y_min, y_max, 0.1))

# Получение предсказаний для каждой точки сетки
```

```
Z = model.predict(poly.transform(np.c_[xx.ravel(), yy.ravel()]))
Z = Z.reshape(xx.shape)

# Построение контурного графика с границей принятия решения
plt.figure(figsize=(10, 6))
plt.contourf(xx, yy, Z, alpha=0.4, cmap='coolwarm')
plt.scatter(X[:, 0], X[:, 1], c=y_pred, cmap='coolwarm', alpha=0.6)
plt.title('Результат классификации для обучающего и тестового подмножеств')
plt.xlabel('X1')
plt.ylabel('X2')
plt.colorbar(label='y')
plt.show()
```



Заключение

Небольшая степень полинома благоприятно сказывается на качестве предсказания, однако, бОльшая степень полинома приводит к усложнению модели и понижает ее эффективность. Как видно на графиках, для отложенной выборки степень полинома от 2 до 10 включительно приводит к хорошей эффективности (для метода кросс-валидации до 5).

Задача 3. Классификация текстовых документов

Вариант 2. Набор SMS сообщений (sms)

3.1 Загрузите исходные данные

In [ ]:
```
# Загружаем исходные данные
df = pd.read_csv("data/SMSSpamCollection", sep='\t', names=['label', 'text'])
df.head(5)
```

Out[ ]:

	label	text
0	ham	Go until jurong point, crazy.. Available only ...
1	ham	Ok lar... Joking wif u oni...
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...
3	ham	U dun say so early hor... U c already then say...
4	ham	Nah I don't think he goes to usf, he lives aro...

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

In [ ]:
```
X = df['text'].to_numpy()
y = df['label'].to_numpy()
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.8, random_state=0)
```

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

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

In [ ]:
```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import StratifiedKFold, GridSearchCV
from sklearn.naive_bayes import BernoulliNB, MultinomialNB
from sklearn.metrics import balanced_accuracy_score
```

In [ ]:
```
# Класс для векторизации текста
countv = CountVectorizer()
# Класс для кросс-валидации
sk = StratifiedKFold(n_splits=4, shuffle=True, random_state=123)
# Переводим обучающую и тестовую выборки в векторное представление
X_train_v = countv.fit_transform(X_train)
X_test_v = countv.transform(X_test)
```

In [ ]:
```
# K-ближайших соседей
# Диапозон параметра количества соседей
range_neighbors = np.arange(1, 150, 20)
# Классифкатор по алгоритму k-ближайших соседей
model_kn = KNeighborsClassifier()
# Класс для поиска гиперпараметров
gs_cv = GridSearchCV(model_kn, param_grid={'n_neighbors' : range_neighbors}, cv=sk, scoring='balanced_accuracy'
# Перебираем все параметры количества соседей
gs_cv.fit(X_train_v, y_train)
# Выводим наилучший
neighbors_best = gs_cv.best_params_['n_neighbors']
neighbors_best
```

Out[ ]:
```
1
```

In [ ]:
```
# Логистическая регрессия
# Диапозон параметров
range_c = np.logspace(-2, 10, 8, base=10)
# Классификатор логистической регресии
model_lr = LogisticRegression()
# Класс для поиска
gs_cv = GridSearchCV(model_lr, param_grid={'C' : range_c}, cv=sk, scoring='balanced_accuracy')
# Перебираем все параметры количества соседей
gs_cv.fit(X_train_v, y_train)
# Выводим наилучший
c_best = gs_cv.best_params_['C']
c_best
```

Out[ ]:
```
193069772.88832456
```

In [ ]:
```
# Бернуллиевский наивный байесовский классификатор
range_alpha_ber = np.logspace(-4, 1, 8, base=10)
# Классификатор
model_ber_nb = BernoulliNB()
# Класс для поиска
gs_cv = GridSearchCV(model_ber_nb, param_grid={'alpha' : range_alpha_ber}, cv=sk, scoring='balanced_accuracy')
# Перебираем все параметры количества соседей
gs_cv.fit(X_train_v, y_train)
# Выводим наилучший
alpha_ber_best = gs_cv.best_params_['alpha']
alpha_ber_best
```

Out[ ]:
```
0.07196856730011521
```

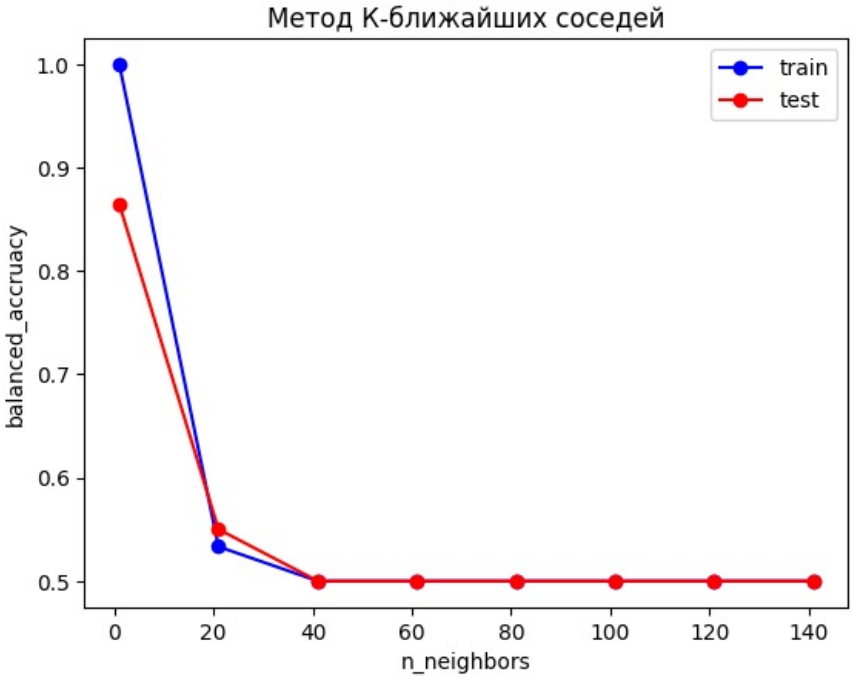
In [ ]:
```
# Мультиномиальный наивный байесовский классификатор
range_alpha_multi = np.logspace(-4, 1, 8, base=10)
# Классификатор
model_multi_nb = MultinomialNB()
# Класс для поиска
gs_cv = GridSearchCV(model_multi_nb, param_grid={'alpha' : range_alpha_multi}, cv=sk, scoring='balanced_accuracy
# Перебираем все параметры количества соседей
```

```
gs_cv.fit(X_train_v, y_train)
# Выводим наилучший
alpha_multi_best = gs_cv.best_params_['alpha']
alpha_multi_best
```

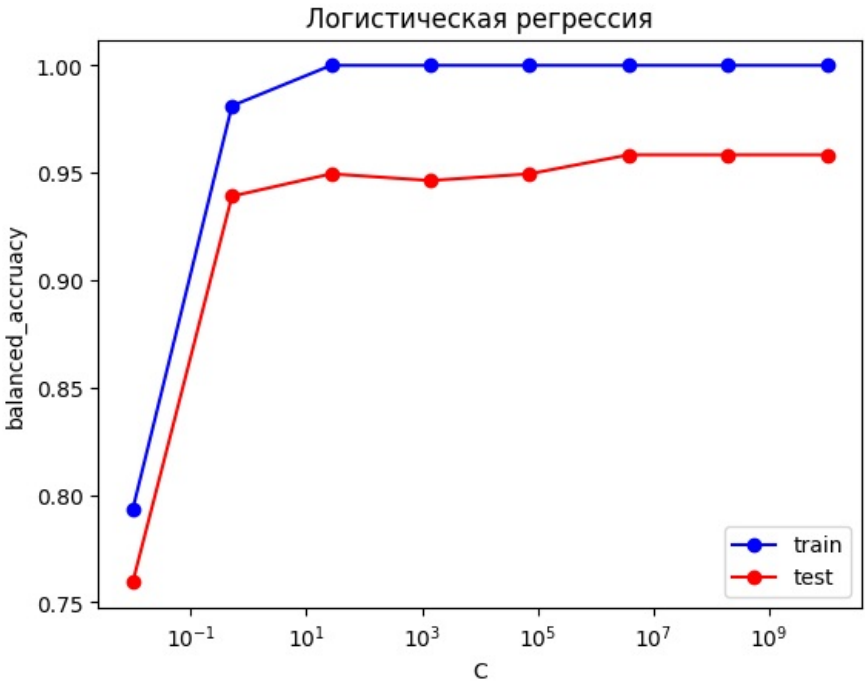
Out[ ]: 0.07196856730011521

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

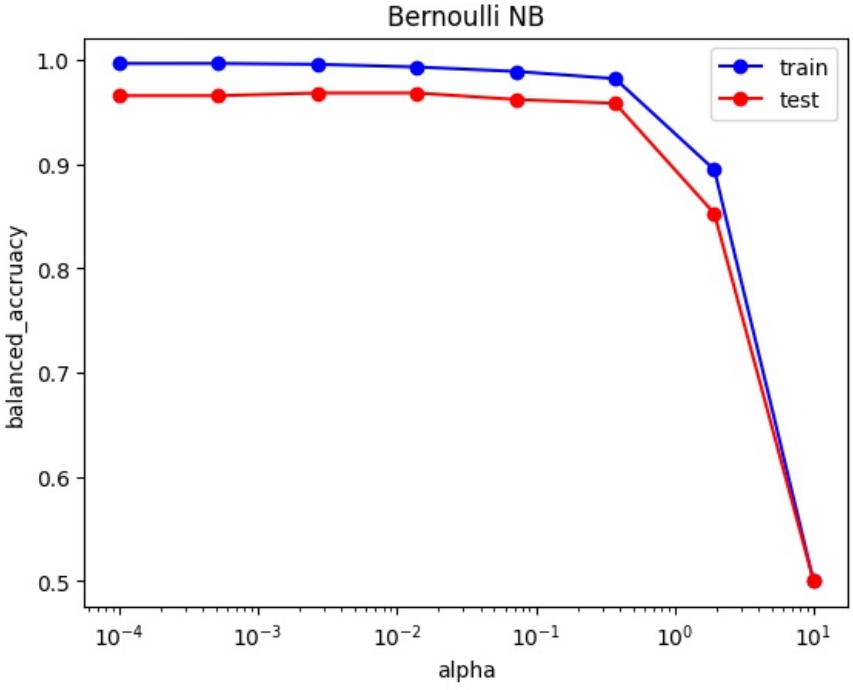
In [ ]:
```
# График для К-ближайших
xx = range_neighbors
yy_train = list()
yy_test = list()
for val in xx:
    model_kn = KNeighborsClassifier(val)
    model_kn.fit(X_train_v, y_train)
    y_train_pred = model_kn.predict(X_train_v)
    y_test_pred = model_kn.predict(X_test_v)
    yy_train.append(balanced_accuracy_score(y_train, y_train_pred))
    yy_test.append(balanced_accuracy_score(y_test, y_test_pred))
plt.title('Метод К-ближайших соседей')
plt.xlabel('n_neighbors')
plt.ylabel('balanced_accruacy')
plt.plot(xx, yy_train, label='train', marker='o', color='blue')
plt.plot(xx, yy_test, label='test', marker='o', color='red')
plt.legend()
plt.show()
```



In [ ]:
```
# График для логистической регрессии
xx = range_c
yy_train = list()
yy_test = list()
for val in xx:
    model_lr = LogisticRegression(C=val)
    model_lr.fit(X_train_v, y_train)
    y_train_pred = model_lr.predict(X_train_v)
    y_test_pred = model_lr.predict(X_test_v)
    yy_train.append(balanced_accuracy_score(y_train, y_train_pred))
    yy_test.append(balanced_accuracy_score(y_test, y_test_pred))
plt.title('Логистическая регрессия')
plt.xlabel('C')
plt.ylabel('balanced_accruacy')
plt.semilogx(xx, yy_train, label='train', marker='o', color='blue')
plt.semilogx(xx, yy_test, label='test', marker='o', color='red')
plt.legend()
plt.show()
```

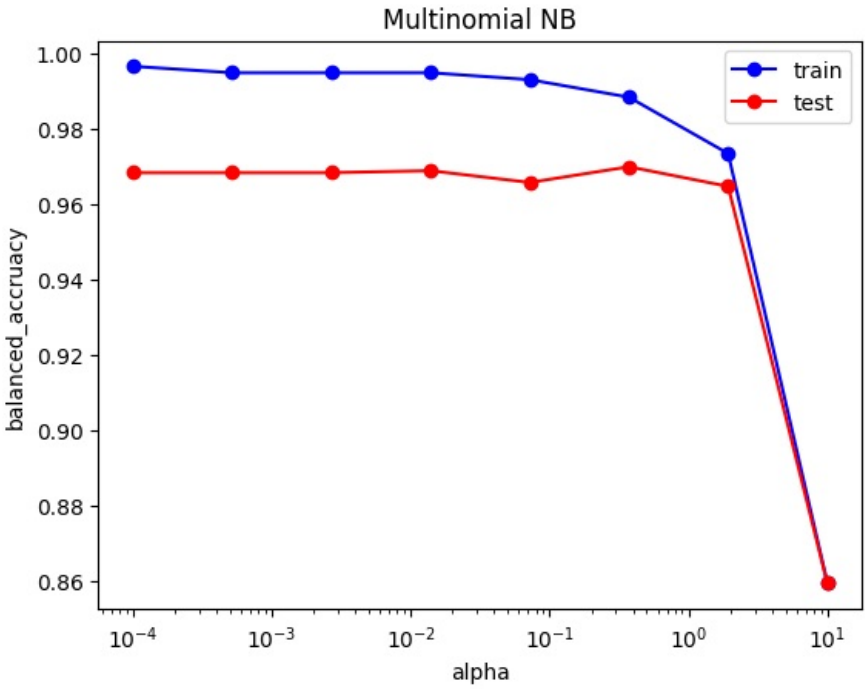


In [ ]:
```
# График для бернуллиевского наивного байесовского классификатора
xx = range_alpha_ber
yy_train = list()
yy_test = list()
for val in xx:
    model_ber_nb = BernoulliNB(alpha=val)
    model_ber_nb.fit(X_train_v, y_train)
    y_train_pred = model_ber_nb.predict(X_train_v)
    y_test_pred = model_ber_nb.predict(X_test_v)
    yy_train.append(balanced_accuracy_score(y_train, y_train_pred))
    yy_test.append(balanced_accuracy_score(y_test, y_test_pred))
plt.title('Bernoulli NB')
plt.xlabel('alpha')
plt.ylabel('balanced_accruacy')
plt.semilogx(xx, yy_train, label='train', marker='o', color='blue')
plt.semilogx(xx, yy_test, label='test', marker='o', color='red')
plt.legend()
plt.show()
```



In [ ]:
```
# График для мультиномиального наивного байесовского классификатора
xx = range_alpha_multi
yy_train = list()
yy_test = list()
for val in xx:
    model_multi_nb = MultinomialNB(alpha=val)
    model_multi_nb.fit(X_train_v, y_train)
    y_train_pred = model_multi_nb.predict(X_train_v)
```

```
    y_test_pred = model_multi_nb.predict(X_test_v)
    yy_train.append(balanced_accuracy_score(y_train, y_train_pred))
    yy_test.append(balanced_accuracy_score(y_test, y_test_pred))
plt.title('Multinomial NB')
plt.xlabel('alpha')
plt.ylabel('balanced_accruacy')
plt.semilogx(xx, yy_train, label='train', marker='o', color='blue')
plt.semilogx(xx, yy_test, label='test', marker='o', color='red')
plt.legend()
plt.show()
```



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 recall_score, precision_score, f1_score
import time
```

In [ ]:
```
# Функция для обучения модели и сбора метрик
def train_model(model, X_train, X_test, y_train, y_test):
    train_start_time = time.time()
    model.fit(X_train, y_train)
    train_time = time.time() - train_start_time
    pred_start_time = time.time()
    y_test_pred = model.predict(X_test)
    pred_time = time.time() - pred_start_time
    accur = balanced_accuracy_score(y_test, y_test_pred)
    r = recall_score(y_test, y_test_pred, pos_label='ham')
    p = precision_score(y_test, y_test_pred, pos_label='ham')
    f1 = f1_score(y_test, y_test_pred, pos_label='ham')
    return accur, r, p, f1, train_time, pred_time
```

In [ ]:
```
# Модель К-ближайших соседей
model_kn = KNeighborsClassifier(n_neighbors=neighbors_best)
accur_kn, r_kn, p_kn, f1_kn, train_time_kn, pred_time_kn = train_model(model_kn, X_train_v, X_test_v, y_train, y
print("Метод К-ближайших соседей")
print("=========================")
print(f"Balanced accuracy: {accur_kn:.4f}")
print(f"Recall score: {r_kn:.4f}")
print(f"Precision score: {r_kn:.4f}")
print(f"F1 score: {r_kn:.4f}")
print(f"Train time: {train_time_kn}")
print(f"Predict time: {pred_time_kn}")
```

```
Метод К-ближайших соседей
=========================
Balanced accuracy: 0.8646
Recall score: 0.9979
Precision score: 0.9979
F1 score: 0.9979
Train time: 0.0029706954956054688
Predict time: 0.15006804466247559
```

In [ ]:
```
# Логистическая регрессия
```

```
model_lr = LogisticRegression(C=c_best)
accur_lr, r_lr, p_lr, f1_lr, train_time_lr, pred_time_lr = train_model(model_lr, X_train_v, X_test_v, y_train, y
print("Логистическая регрессия")
print("========================")
print(f"Balanced accuracy: {accur_lr:.4f}")
print(f"Recall score: {r_lr:.4f}")
print(f"Precision score: {r_lr:.4f}")
print(f"F1 score: {r_lr:.4f}")
print(f"Train time: {train_time_lr}")
print(f"Predict time: {pred_time_lr}")
```

```
Логистическая регрессия
========================
Balanced accuracy: 0.9583
Recall score: 0.9979
Precision score: 0.9979
F1 score: 0.9979
Train time: 0.04702639579772949
Predict time: 0.0001876354217529297
```

In [ ]:
```
# Бернуллиевский наивный байесовский классификатор
model_ber_nb = BernoulliNB(alpha=alpha_ber_best)
accur_ber, r_ber, p_ber, f1_ber, train_time_ber, pred_time_ber = train_model(model_ber_nb, X_train_v, X_test_v,
print("Бернуллиевский НБК")
print("==================")
print(f"Balanced accuracy: {accur_ber:.4f}")
print(f"Recall score: {r_ber:.4f}")
print(f"Precision score: {r_ber:.4f}")
print(f"F1 score: {r_ber:.4f}")
print(f"Train time: {train_time_ber}")
print(f"Predict time: {pred_time_ber}")
```

```
Бернуллиевский НБК
==================
Balanced accuracy: 0.9620
Recall score: 0.9990
Precision score: 0.9990
F1 score: 0.9990
Train time: 0.008309364318847656
Predict time: 0.0008988380432128906
```

In [ ]:
```
# Мультиномиальный наивный байесовский классификатор
model_multi_nb = MultinomialNB(alpha=alpha_multi_best)
accur_multi, r_multi, p_multi, f1_multi, train_time_multi, pred_time_multi = train_model(model_multi_nb, X_trai
print("Мультиномиальный НБК")
print("====================")
print(f"Balanced accuracy: {accur_multi:.4f}")
print(f"Recall score: {r_multi:.4f}")
print(f"Precision score: {r_multi:.4f}")
print(f"F1 score: {r_multi:.4f}")
print(f"Train time: {train_time_multi}")
print(f"Predict time: {pred_time_multi}")
```

```
Мультиномиальный НБК
====================
Balanced accuracy: 0.9656
Recall score: 0.9937
Precision score: 0.9937
F1 score: 0.9937
Train time: 0.008083343505859375
Predict time: 0.0002541542053222656
```

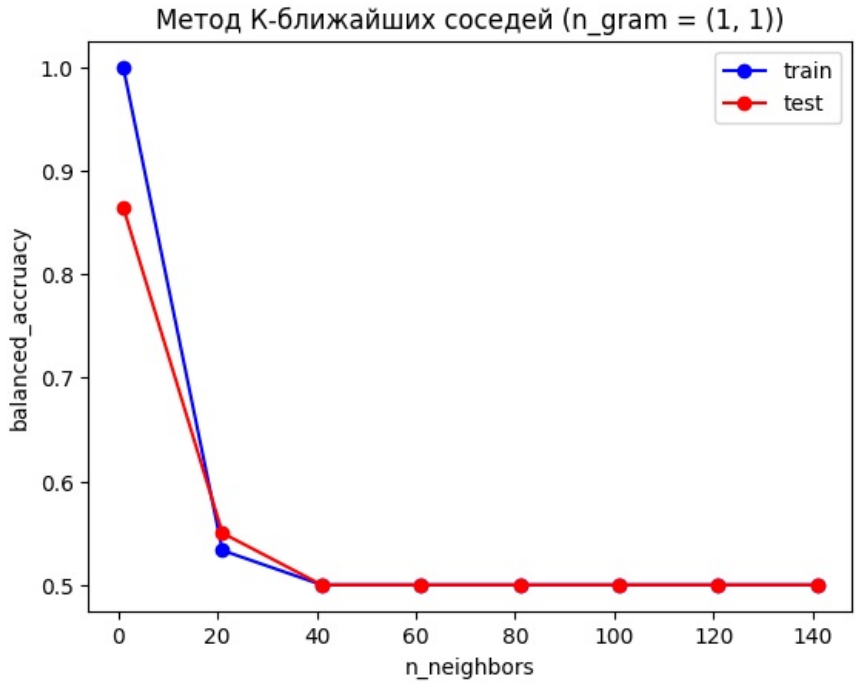
3.6 Выполните пункты 3-5 для `n-gram=1`, `n-gram=2` и `n-gram=(1,2)`

In [ ]:
```
# n-gram параметры
n_grams = [(1, 1), (2, 2), (1, 2)]
# датафрейм с результатами для пункта 8
res_df = pd.DataFrame(columns=['Метод', 'n_gram', 'Параметры модели', 'Время обучения', 'Время предсказания', '
```

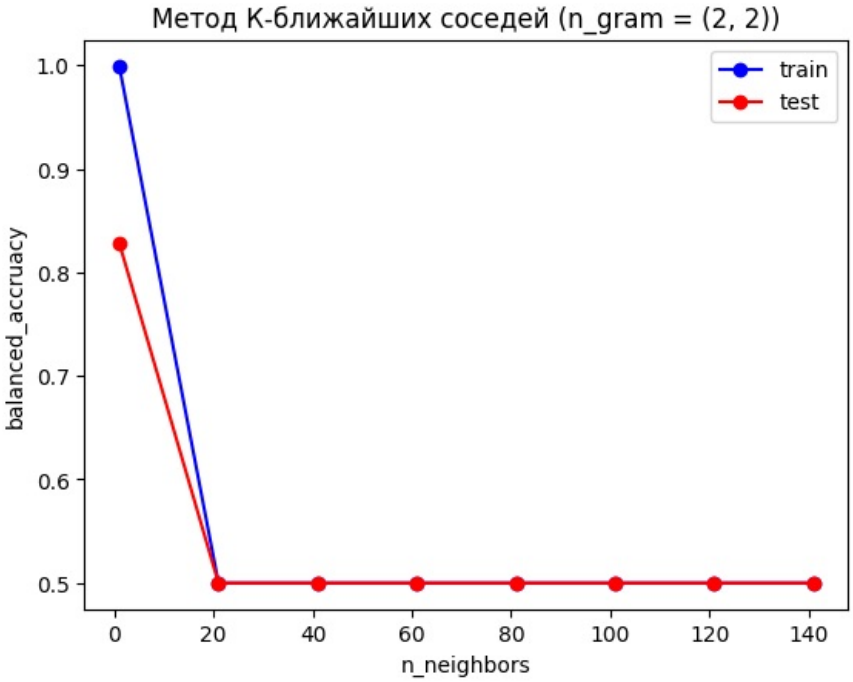
In [ ]:
```
# Для К-ближайших
for n_gram in n_grams:
    # Класс для векторизации текста
    countv = CountVectorizer(ngram_range=n_gram)
    X_train_v = countv.fit_transform(X_train)
    X_test_v = countv.transform(X_test)
    # Находим наилучшие параметры
    range_neighbors = np.arange(1, 150, 20)
    model_kn = KNeighborsClassifier()
    gs_cv = GridSearchCV(model_kn, param_grid={'n_neighbors' : range_neighbors}, cv=sk, scoring='balanced_accura
    gs_cv.fit(X_train_v, y_train)
    print(f"Для n_gram {n_gram} лучший параметр n_neighbors: {gs_cv.best_params_['n_neighbors']}")
    print("========================")
```

```python
    # Собираем метрики
    model_kn = KNeighborsClassifier(n_neighbors=gs_cv.best_params_['n_neighbors'])
    accur_kn, r_kn, p_kn, f1_kn, train_time_kn, pred_time_kn = train_model(model_kn, X_train_v, X_test_v, y_tra
    print(f"Метод K-ближайших соседей (n_gram = {n_gram})")
    print("=========================")
    print(f"Balanced accuracy: {accur_kn:.4f}")
    print(f"Recall score: {r_kn:.4f}")
    print(f"Precision score: {r_kn:.4f}")
    print(f"F1 score: {r_kn:.4f}")
    print(f"Train time: {train_time_kn}")
    print(f"Predict time: {pred_time_kn}")

    # Строим график
    xx = range_neighbors
    yy_train = list()
    yy_test = list()
    for val in xx:
        model_kn = KNeighborsClassifier(n_neighbors=val)
        model_kn.fit(X_train_v, y_train)
        y_train_pred = model_kn.predict(X_train_v)
        y_test_pred = model_kn.predict(X_test_v)
        yy_train.append(balanced_accuracy_score(y_train, y_train_pred))
        yy_test.append(balanced_accuracy_score(y_test, y_test_pred))
    plt.title(f'Метод K-ближайших соседей (n_gram = {n_gram})')
    plt.xlabel('n_neighbors')
    plt.ylabel('balanced_accruacy')
    plt.plot(xx, yy_train, label='train', marker='o', color='blue')
    plt.plot(xx, yy_test, label='test', marker='o', color='red')
    plt.legend()
    plt.show()

    # Сохраняем результаты в датафрейм
    res_df.loc[len(res_df)] = ['Метод K-ближайших соседей', n_gram, gs_cv.best_params_['n_neighbors'], train_ti
```

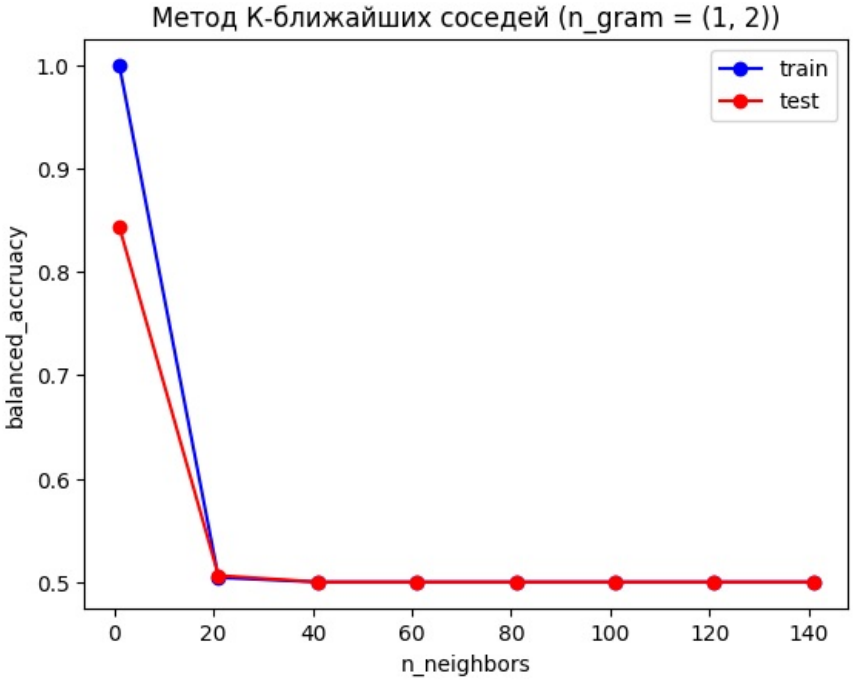
```
Для n_gram (1, 1) лучший параметр n_neighbors: 1
=========================
Метод K-ближайших соседей (n_gram = (1, 1))
=========================
Balanced accuracy: 0.8646
Recall score: 0.9979
Precision score: 0.9979
F1 score: 0.9979
Train time: 0.002554178237915039
Predict time: 0.12397885322570801
```




```
Для n_gram (2, 2) лучший параметр n_neighbors: 1
=========================
Метод K-ближайших соседей (n_gram = (2, 2))
=========================
Balanced accuracy: 0.8281
Recall score: 1.0000
Precision score: 1.0000
F1 score: 1.0000
Train time: 0.0032749176025390625
Predict time: 0.08379149436950684
```



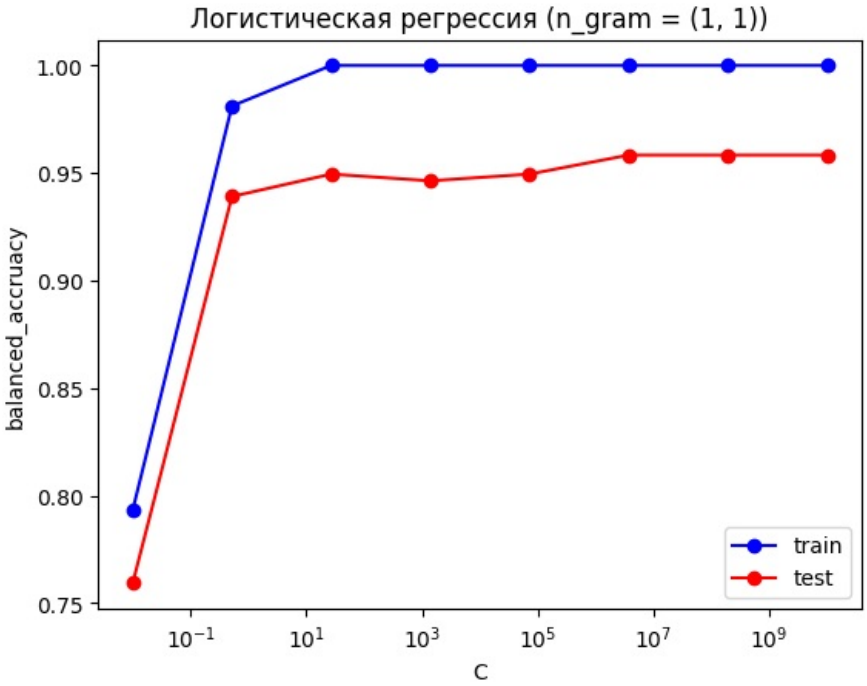

```
Для n_gram (1, 2) лучший параметр n_neighbors: 1
=========================
Метод K-ближайших соседей (n_gram = (1, 2))
=========================
Balanced accuracy: 0.8438
Recall score: 1.0000
Precision score: 1.0000
F1 score: 1.0000
Train time: 0.0035092830657958984
Predict time: 0.14029574394226074
```



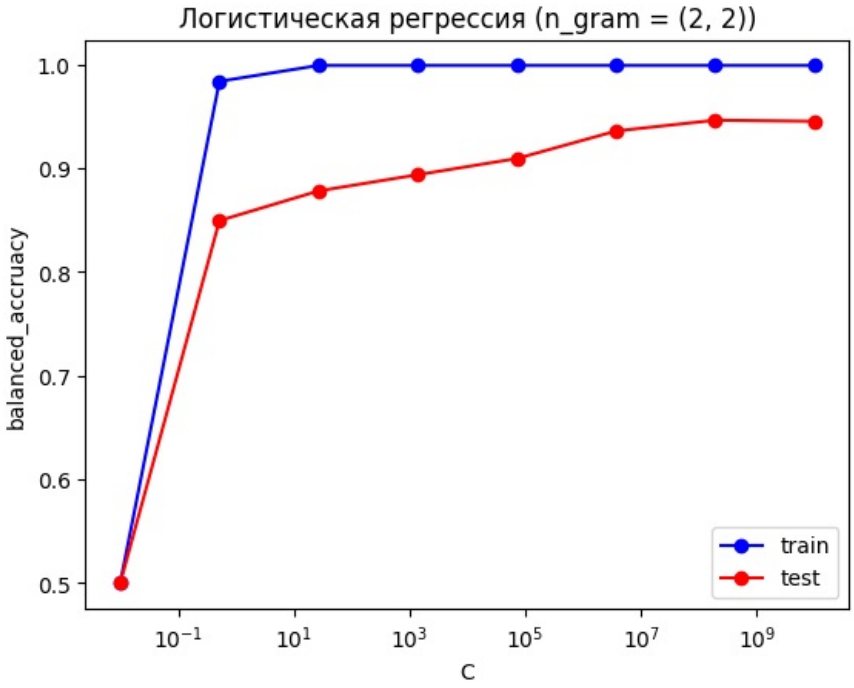
In [ ]:
```python
# Для логистической регресси
for n_gram in n_grams:
    # Класс для векторизации текста
    countv = CountVectorizer(ngram_range=n_gram)
    X_train_v = countv.fit_transform(X_train)
    X_test_v = countv.transform(X_test)
    # Находим наилучшие параметры
    range_lr = np.logspace(-2, 10, 8, base=10)
    model_lr = LogisticRegression()
    gs_cv = GridSearchCV(model_lr, param_grid={'C' : range_lr}, cv=sk, scoring='balanced_accuracy')
    gs_cv.fit(X_train_v, y_train)
    print(f"Для n_gram {n_gram} лучший параметр C: {gs_cv.best_params_['C']}")
    print("=========================")

    # Собираем метрики
```

```python
    model_lr = LogisticRegression(C=gs_cv.best_params_['C'])
    accur_lr, r_lr, p_lr, f1_lr, train_time_lr, pred_time_lr = train_model(model_lr, X_train_v, X_test_v, y_tra
    print(f"Метод логистической регрессии (n_gram = {n_gram})")
    print("=========================")
    print(f"Balanced accuracy: {accur_lr:.4f}")
    print(f"Recall score: {r_lr:.4f}")
    print(f"Precision score: {r_lr:.4f}")
    print(f"F1 score: {r_lr:.4f}")
    print(f"Train time: {train_time_lr}")
    print(f"Predict time: {pred_time_lr}")

    # Строим график
    xx = range_c
    yy_train = list()
    yy_test = list()
    for val in xx:
        model_lr = LogisticRegression(C=val)
        model_lr.fit(X_train_v, y_train)
        y_train_pred = model_lr.predict(X_train_v)
        y_test_pred = model_lr.predict(X_test_v)
        yy_train.append(balanced_accuracy_score(y_train, y_train_pred))
        yy_test.append(balanced_accuracy_score(y_test, y_test_pred))
    plt.title(f"Логистическая регрессия (n_gram = {n_gram})")
    plt.xlabel('C')
    plt.ylabel('balanced_accruacy')
    plt.semilogx(xx, yy_train, label='train', marker='o', color='blue')
    plt.semilogx(xx, yy_test, label='test', marker='o', color='red')
    plt.legend()
    plt.show()

    # Сохраняем результаты в датафрейм
    res_df.loc[len(res_df)] = ['Логистическая регрессия', n_gram, gs_cv.best_params_['C'], train_time_lr, pred_
```

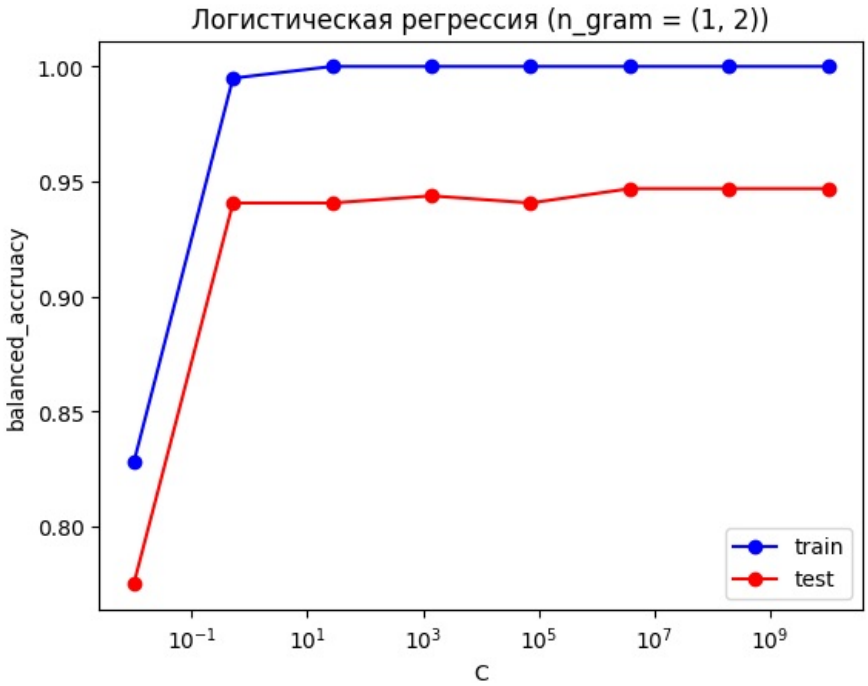
```
Для n_gram (1, 1) лучший параметр C: 193069772.88832456
=========================
Метод логистической регрессии (n_gram = (1, 1))
=========================
Balanced accuracy: 0.9583
Recall score: 0.9979
Precision score: 0.9979
F1 score: 0.9979
Train time: 0.040482282638549805
Predict time: 0.0002799034118652344
```




```
Для n_gram (2, 2) лучший параметр C: 193069772.88832456
=========================
Метод логистической регрессии (n_gram = (2, 2))
=========================
Balanced accuracy: 0.9463
Recall score: 0.9927
Precision score: 0.9927
F1 score: 0.9927
Train time: 0.4650125503540039
Predict time: 0.00023174285888671875
```



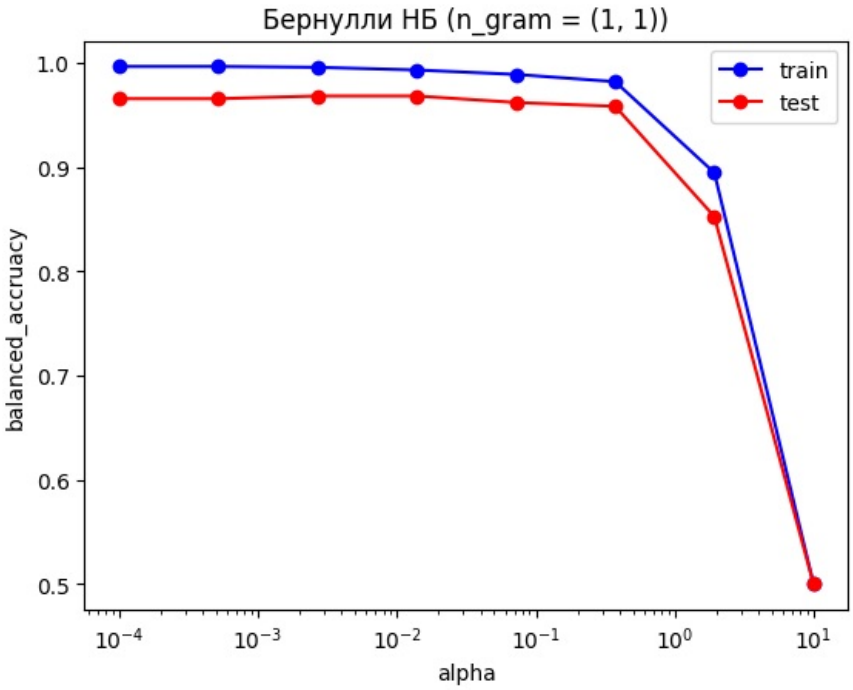

```
Для n_gram (1, 2) лучший параметр C: 3727593.720314938
=========================
Метод логистической регрессии (n_gram = (1, 2))
=========================
Balanced accuracy: 0.9469
Recall score: 1.0000
Precision score: 1.0000
F1 score: 1.0000
Train time: 0.5255286693572998
Predict time: 0.0003197193145751953
```



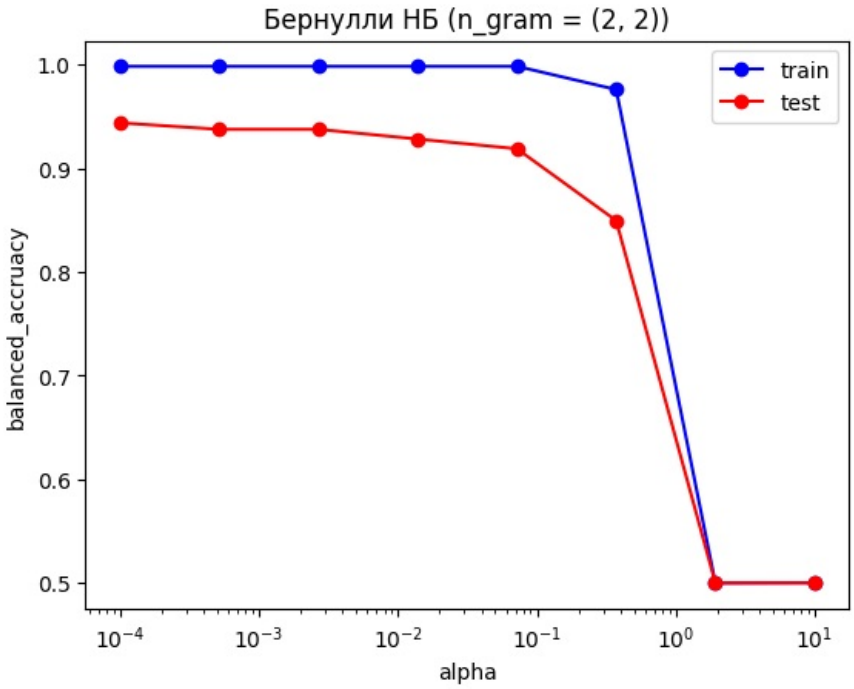
In [ ]:
```python
# Для Бернулли НБ
for n_gram in n_grams:
    # Класс для векторизации текста
    countv = CountVectorizer(ngram_range=n_gram)
    X_train_v = countv.fit_transform(X_train)
    X_test_v = countv.transform(X_test)
    # Находим наилучшие параметры
    range_alpha_ber = np.logspace(-4, 1, 8, base=10)
    model_ber_nb = BernoulliNB()
    gs_cv = GridSearchCV(model_ber_nb, param_grid={'alpha' : range_alpha_ber}, cv=sk, scoring='balanced_accuracy
    gs_cv.fit(X_train_v, y_train)
    print(f"Для n_gram {n_gram} лучший параметр alpha: {gs_cv.best_params_['alpha']}")
    print("=========================")
```

```python
    # Собираем метрики
    model_ber_nb = BernoulliNB(alpha=gs_cv.best_params_['alpha'])
    accur_ber, r_ber, p_ber, f1_ber, train_time_ber, pred_time_ber = train_model(model_ber_nb, X_train_v, X_tes
    print(f"Метод Бернулли НБ (n_gram = {n_gram})")
    print("=========================")
    print(f"Balanced accuracy: {accur_ber:.4f}")
    print(f"Recall score: {r_ber:.4f}")
    print(f"Precision score: {r_ber:.4f}")
    print(f"F1 score: {r_ber:.4f}")
    print(f"Train time: {train_time_ber}")
    print(f"Predict time: {pred_time_ber}")

    # Строим график
    xx = range_alpha_ber
    yy_train = list()
    yy_test = list()
    for val in xx:
        model_multi_nb = BernoulliNB(alpha=val)
        model_multi_nb.fit(X_train_v, y_train)
        y_train_pred = model_multi_nb.predict(X_train_v)
        y_test_pred = model_multi_nb.predict(X_test_v)
        yy_train.append(balanced_accuracy_score(y_train, y_train_pred))
        yy_test.append(balanced_accuracy_score(y_test, y_test_pred))
    plt.title(f"Бернулли НБ (n_gram = {n_gram})")
    plt.xlabel('alpha')
    plt.ylabel('balanced_accruacy')
    plt.semilogx(xx, yy_train, label='train', marker='o', color='blue')
    plt.semilogx(xx, yy_test, label='test', marker='o', color='red')
    plt.legend()
    plt.show()

    # Сохраняем результаты в датафрейм
    res_df.loc[len(res_df)] = ['Бернулли НБ', n_gram, gs_cv.best_params_['alpha'], train_time_ber, pred_time_be
```

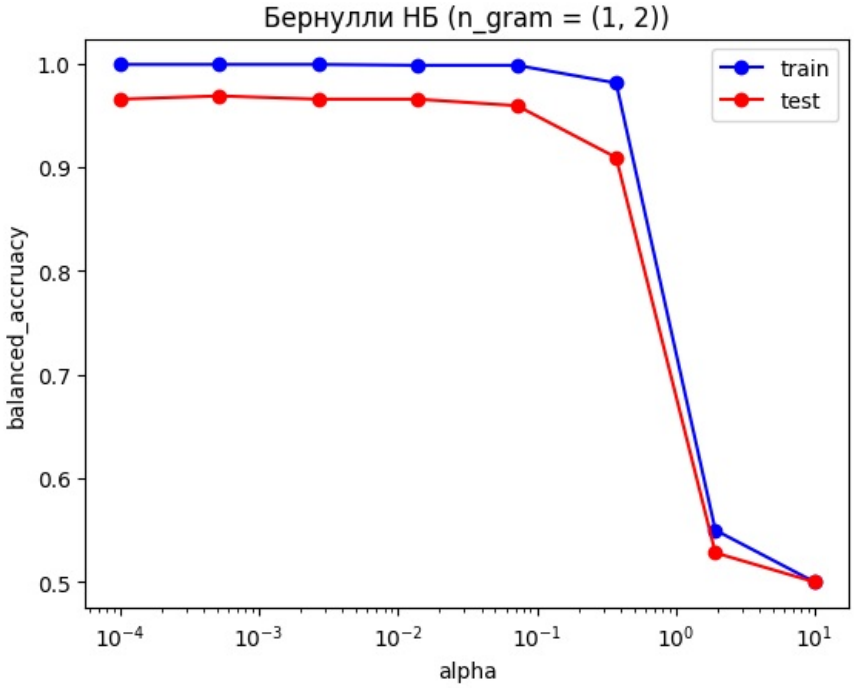
```
Для n_gram (1, 1) лучший параметр alpha: 0.07196856730011521
=========================
Метод Бернулли НБ (n_gram = (1, 1))
=========================
Balanced accuracy: 0.9620
Recall score: 0.9990
Precision score: 0.9990
F1 score: 0.9990
Train time: 0.007494211196899414
Predict time: 0.0008742809295654297
```




```
Для n_gram (2, 2) лучший параметр alpha: 0.0001
=========================
Метод Бернулли НБ (n_gram = (2, 2))
=========================
Balanced accuracy: 0.9437
Recall score: 1.0000
Precision score: 1.0000
F1 score: 1.0000
Train time: 0.01071929931640625
Predict time: 0.0018019676208496094
```



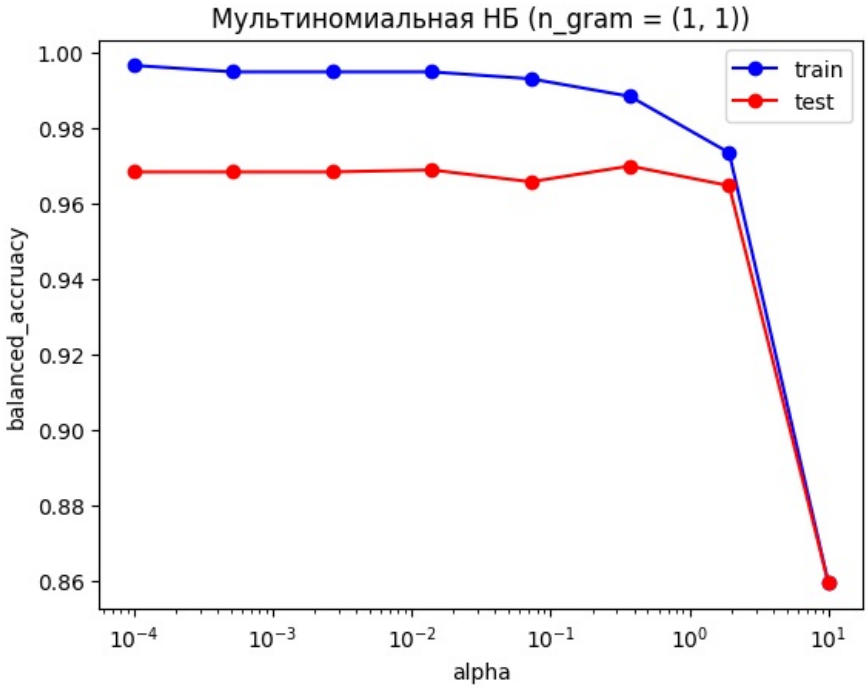

```
Для n_gram (1, 2) лучший параметр alpha: 0.0005179474679231213
=========================
Метод Бернулли НБ (n_gram = (1, 2))
=========================
Balanced accuracy: 0.9688
Recall score: 1.0000
Precision score: 1.0000
F1 score: 1.0000
Train time: 0.009244441986083984
Predict time: 0.0023872852325439453
```



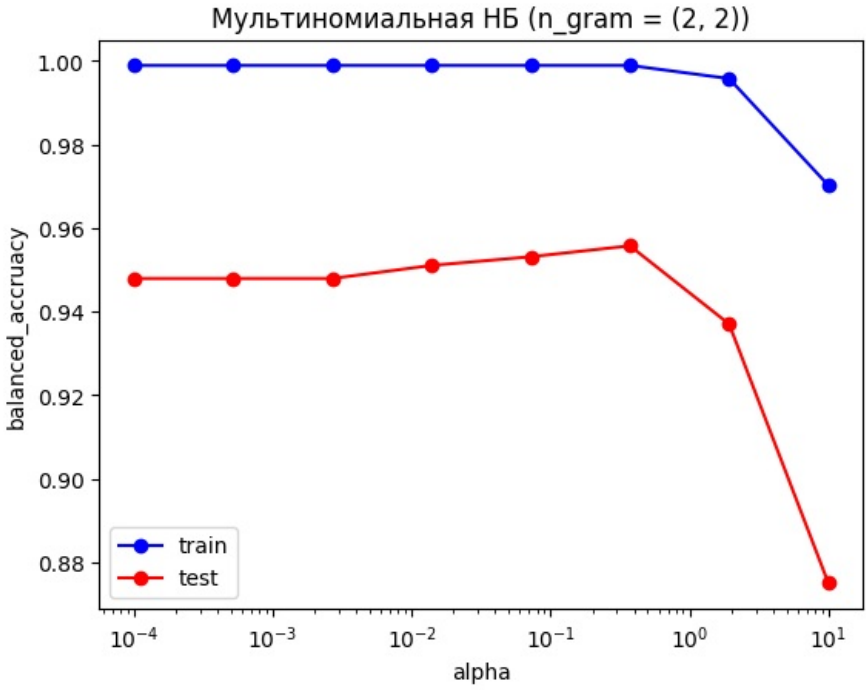
In [ ]:
```python
# Для Мультномиальной НБ
for n_gram in n_grams:
    # Класс для векторизации текста
    countv = CountVectorizer(ngram_range=n_gram)
    X_train_v = countv.fit_transform(X_train)
    X_test_v = countv.transform(X_test)
    # Находим наилучшие параметры
    range_alpha_multi = np.logspace(-4, 1, 8, base=10)
    model_multi_nb = MultinomialNB()
    gs_cv = GridSearchCV(model_multi_nb, param_grid={'alpha' : range_alpha_multi}, cv=sk, scoring='balanced_acc
    gs_cv.fit(X_train_v, y_train)
    print(f"Для n_gram {n_gram} лучший параметр alpha: {gs_cv.best_params_['alpha']}")
    print("=========================")

    # Собираем метрики
```

```
    model_multi_nb = MultinomialNB(alpha=gs_cv.best_params_['alpha'])
    accur_multi, r_multi, p_multi, f1_multi, train_time_multi, pred_time_multi = train_model(model_multi_nb, X_
    print(f"Метод Бернулли НБ (n_gram = {n_gram})")
    print("=========================")
    print(f"Balanced accuracy: {accur_multi:.4f}")
    print(f"Recall score: {r_multi:.4f}")
    print(f"Precision score: {r_multi:.4f}")
    print(f"F1 score: {r_multi:.4f}")
    print(f"Train time: {train_time_multi}")
    print(f"Predict time: {pred_time_multi}")

    # Строим график
    xx = range_alpha_multi
    yy_train = list()
    yy_test = list()
    for val in xx:
        model_multi_nb = MultinomialNB(alpha=val)
        model_multi_nb.fit(X_train_v, y_train)
        y_train_pred = model_multi_nb.predict(X_train_v)
        y_test_pred = model_multi_nb.predict(X_test_v)
        yy_train.append(balanced_accuracy_score(y_train, y_train_pred))
        yy_test.append(balanced_accuracy_score(y_test, y_test_pred))
    plt.title(f"Мультиномиальная НБ (n_gram = {n_gram})")
    plt.xlabel('alpha')
    plt.ylabel('balanced_accruacy')
    plt.semilogx(xx, yy_train, label='train', marker='o', color='blue')
    plt.semilogx(xx, yy_test, label='test', marker='o', color='red')
    plt.legend()
    plt.show()

    # Сохраняем результаты в датафрейм
    res_df.loc[len(res_df)] = ['Мультномиальная НБ', n_gram, gs_cv.best_params_['alpha'], train_time_multi, pre
```

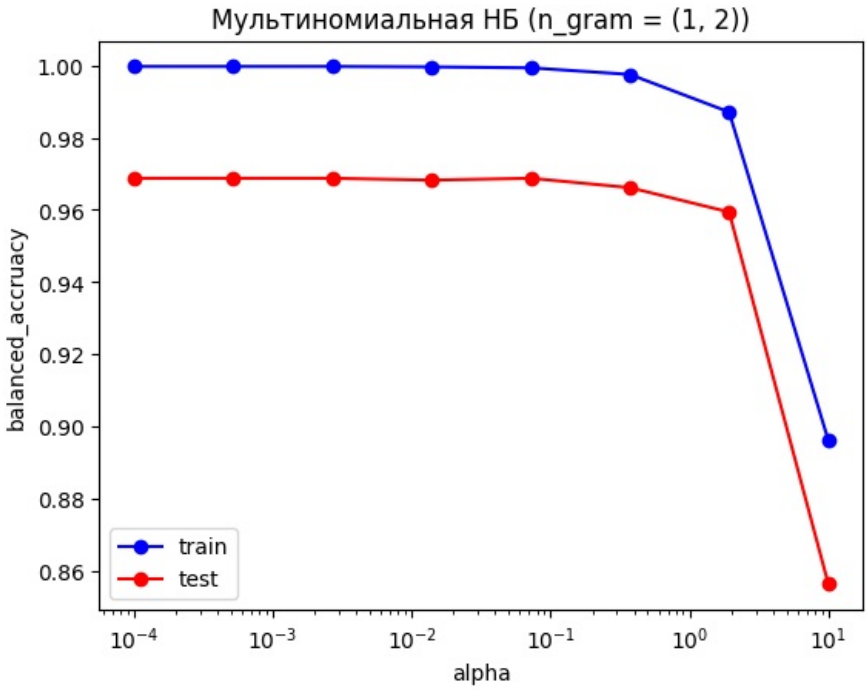
```
Для n_gram (1, 1) лучший параметр alpha: 0.07196856730011521
=========================
Метод Бернулли НБ (n_gram = (1, 1))
=========================
Balanced accuracy: 0.9656
Recall score: 0.9937
Precision score: 0.9937
F1 score: 0.9937
Train time: 0.006832122802734375
Predict time: 0.00019788742065429688
```




```
Для n_gram (2, 2) лучший параметр alpha: 1.9306977288832496
=========================
Метод Бернулли НБ (n_gram = (2, 2))
=========================
Balanced accuracy: 0.9370
Recall score: 0.9990
Precision score: 0.9990
F1 score: 0.9990
Train time: 0.007555961608886719
Predict time: 0.00040841102600097656
```




```
Для n_gram (1, 2) лучший параметр alpha: 0.0001
=========================
Метод Бернулли НБ (n_gram = (1, 2))
=========================
Balanced accuracy: 0.9687
Recall score: 0.9937
Precision score: 0.9937
F1 score: 0.9937
Train time: 0.01010274887084961
Predict time: 0.0005066394805908203
```



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`))

In [ ]:
```
res_df
```

Out[ ]:

	Метод	n_gram	Параметры модели	Время обучения	Время предсказания	Точность	R	P	F1
0	Метод K-ближайших соседей	(1, 1)	1.000000e+00	0.002554	0.123979	0.864578	0.997906	0.956827	0.976935
1	Метод K-ближайших соседей	(2, 2)	1.000000e+00	0.003275	0.083791	0.828125	1.000000	0.945545	0.972010
2	Метод K-ближайших соседей	(1, 2)	1.000000e+00	0.003509	0.140296	0.843750	1.000000	0.950249	0.974490
3	Логистическая регрессия	(1, 1)	1.930698e+08	0.040482	0.000280	0.958328	0.997906	0.986542	0.992192
4	Логистическая регрессия	(2, 2)	1.930698e+08	0.465013	0.000232	0.946335	0.992670	0.983402	0.988015
5	Логистическая регрессия	(1, 2)	3.727594e+06	0.525529	0.000320	0.946875	1.000000	0.982510	0.991178
6	Бернулли НБ	(1, 1)	7.196857e-02	0.007494	0.000874	0.961976	0.998953	0.987578	0.993233
7	Бернулли НБ	(2, 2)	1.000000e-04	0.010719	0.001802	0.943750	1.000000	0.981501	0.990664
8	Бернулли НБ	(1, 2)	5.179475e-04	0.009244	0.002387	0.968750	1.000000	0.989637	0.994792
9	Мультномиальная НБ	(1, 1)	7.196857e-02	0.006832	0.000198	0.965609	0.993717	0.989572	0.991641
10	Мультномиальная НБ	(2, 2)	1.930698e+00	0.007556	0.000408	0.936976	0.998953	0.979466	0.989114
11	Мультномиальная НБ	(1, 2)	1.000000e-04	0.010103	0.000507	0.968734	0.993717	0.990605	0.992159

3.8 Сделайте выводы по полученным результатам (преимущества и недостатки методов)

По результатам проделанной работы можно сделать следующие выводы: метод KNN в сравнении с другими обладает невысокой точностью и требует большего количества времени предсказание. Логистическая регрессия, наоборот, имеет высокую точность и быстро предсказывает, однако для обучения требует больше времени (особенно это видно на биграммах). Наивные байесовские классификаторы показывают самую высокую точность среди приведенных моделей. Таким образом, если нужна быстрая обучаемость, метод KNN будет неплохим вариантом, для быстрых предсказаний подходит логистическая регрессия, а если необходимо добиться наилучшей точности, то наивные байесовские классификаторы будут подходящим вариантом