Липецкий государственный технический университет

Факультет автоматизации и информатики

Кафедра автоматизированных систем управления

ЛАБОРАТОРНАЯ РАБОТА №1

по дисциплине «Прикладные интеллектуальные системы и экспертные системы»

Бинарная классификация фактографических данных

Студент Посаднев В.В.

Группа М-ИАП-22-1

Руководитель Кургасов В.В.

Доцент

Липецк 2022 г.

Задание кафедры

1) В среде Jupyter Notebook создать новый ноутбук (Notebook)

2) Импортировать необходимые для работы библиотеки и модули

3) Загрузить данные в соответствии с вариантом

4) Вывести первые 15 элементов выборки (координаты точек и метки класса)

5) Отобразить на графике сгенерированную выборку. Объекты разных классов должны иметь разные цвета.

6) Разбить данные на обучающую (train) и тестовую (test) выборки в пропорции 75% - 25% соответственно.

7) Отобразить на графике обучающую и тестовую выборки. Объекты разных классов должны иметь разные цвета.

8) Реализовать модели классификаторов, обучить их на обучающем множестве. Применить модели на тестовой выборке, вывести результаты классификации:

- Истинные и предсказанные метки классов
- Матрицу ошибок (confusion matrix)
- Значение полноты, точности, f1-меры и аккуратности
- Значение площади под кривой ошибок (AUC ROC)
- Отобразить на графике область принятия решений по каждому классу

В качестве методов классификации использовать:

- Метод к-ближайших соседей (n_neighbors = {1, 3, 5, 9})
- Наивный байесовский метод
- Случайный лес (n_estimators = {5, 10, 15, 20, 50})

9) По каждому пункту работы занести в отчет программный код и результат вывода.

10) По результатам п.8 занести в отчет таблицу с результатами классификации всеми методами и выводы о наиболее подходящем методе классификации ваших данных.

11) Изучить, как изменится качестве классификации, если на тестовую часть выделить 10% выборки, 35% выборки. Для этого повторить п.п. 6 – 10.

Вариант №11

Вид классов: classification

random_state: 15

class_sep: 0.6

n_features: 2

n_redundant: 0

n_informative: 1

n_clusters_per_class: 1

Оглавление

Ход работы

Подготовка данных

Для генерации данных воспользуемся функцией make_classification из пакета sklearn.datasets. Результат генерации данных и вывод первых 15 значений представлен на рисунке 1.

```
In 4
X, y = make_classification(n_features=2,
                           n_samples=1000,
                           n_redundant=0,
                           n_informative=1,
                           n_clusters_per_class=1,
                           random_state=15,
                           class_sep=0.6)
```

```
In 5
print('Координаты точек: ')
print(X[:15])
print('Метки класса: ')
print(y[:15])
```

```
Координаты точек:
[[-0.32654509 -0.48287283]
 [-0.56423228  0.36908979]
 [ 1.80734839  0.64084024]
 [-1.13815022 -0.3922336 ]
 [-0.77269253  0.98787649]
 [-0.76362783 -1.03345078]
 [ 1.27084064  1.02090267]
 [ 0.28768416  0.02922487]
 [-0.19381938 -1.04395297]
 [-0.8936574  -0.64384405]
 [ 0.30631716  0.81656104]
 [-1.64047657  0.40696626]
 [-0.49234077 -1.04988151]
 [ 1.17360256 -0.58037911]
 [-0.66270457 -0.25318302]]
Метки класса:
[0 1 1 0 1 0 1 1 0 0 1 1 0 0 0]
```

Рисунок 1 – Генерация данных и вывод первых 15-ти результатов

Для отображения на графике сгенерированной выборки с выделением классов разными цветами воспользуемся функцией scatter из библиотеки matplotlib.pyplot. Результат визуализации данных представлен на рисунке 2.

```
plt.scatter(X[:, 0], X[:, 1], c=y)
plt.show()
```

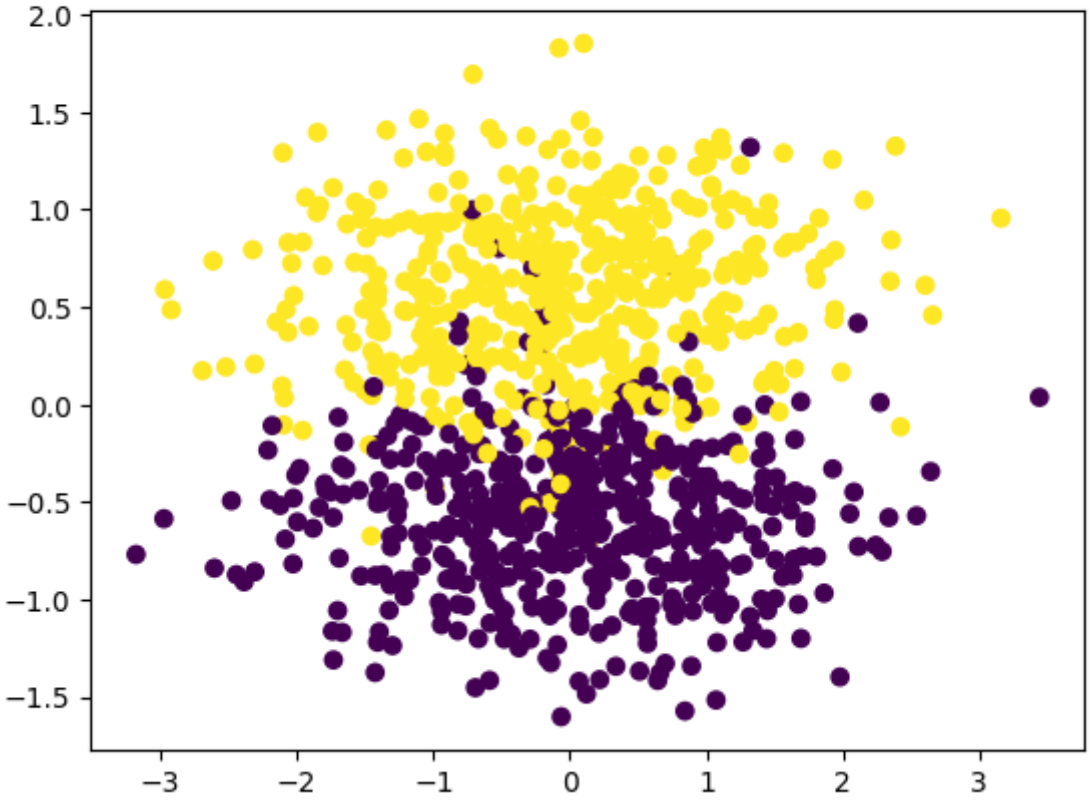

Рисунок 2 – Визуализация сгенерированных данных

Для разделения данных на обучающую и тестовую выборку воспользуемся функцией train_test_split из пакета sklearn.model_selection. Скрипт для разделения данных представлен на рисунке 3. Результат разбиения выборки на тестовую и обучающую с последующей их визуализацией представлены на рисунке 4 и 5.

```
X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    test_size=0.25,
                                                    random_state=42)
```

Рисунок 3 – Код для разделения выборки

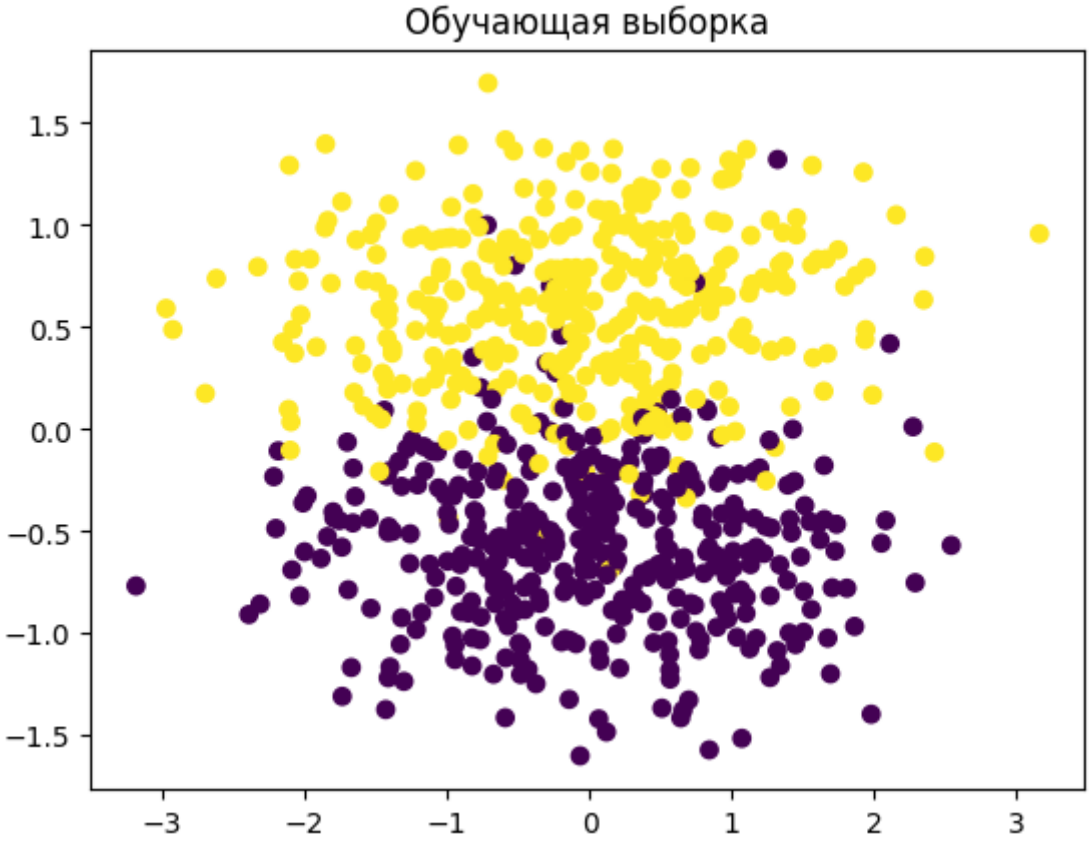

Рисунок 4 – Визуализация обучающей выборки

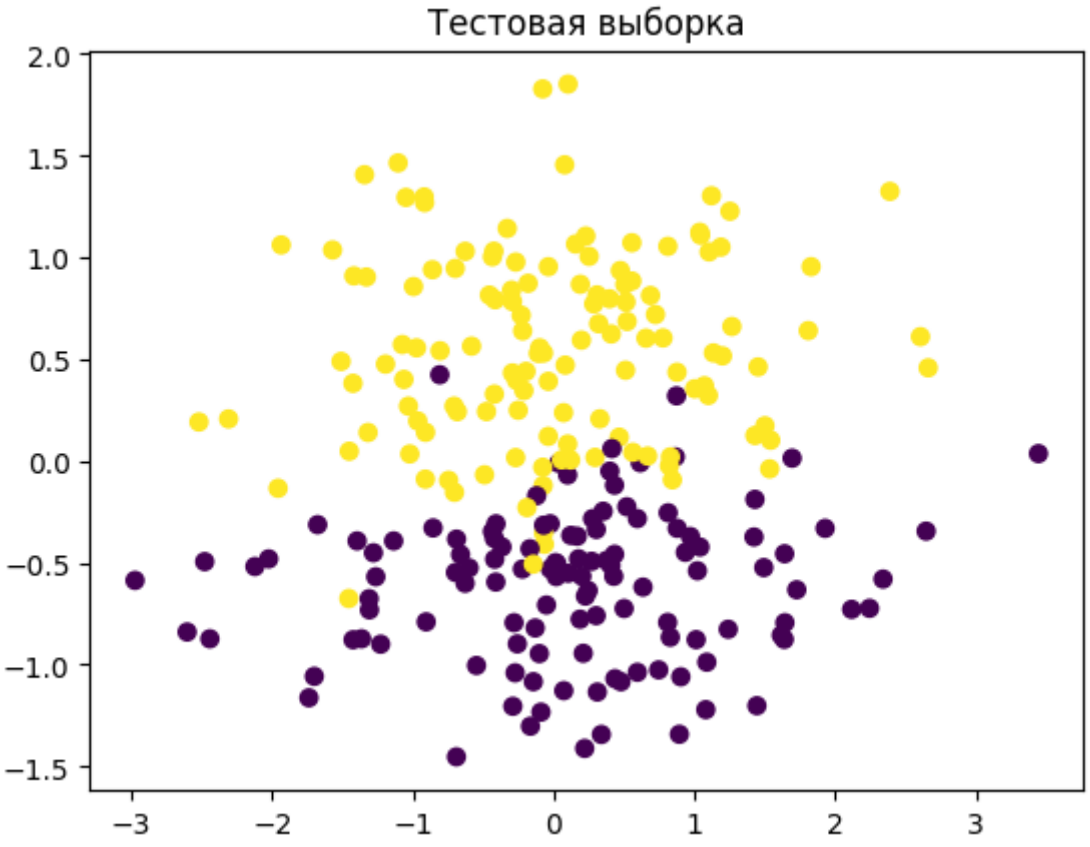

Рисунок 5 – Визуализация тестовой выборки

Классификация с помощью метода к-ближайших соседей

Использование параметра n_neighbors = 1. Составленный код для использования данного классификатора с данным параметром представлен на рисунке 6. Полученная информация о точности классификации при использовании данного метода представлена на рисунке 7.

```
knn = KNeighborsClassifier(n_neighbors=1, metric='euclidean')

# Обучаем модель данных
knn.fit(X_train, y_train)

# Оцениваем качество модели
prediction = knn.predict(X_test)

# Выводим сводную информацию
show_info(knn, 'ближайшие соседи (1)', y_test, prediction)
```

Рисунок 6 – Код для классификатора с помощью метода к-ближайших соседей с параметром n_neighbors = 1

```
Матрица неточностей
[[113   8]
 [ 20 109]]

Точность классификации: 0.888

Полнота:
              precision    recall  f1-score   support

           0       0.85      0.93      0.89       121
           1       0.93      0.84      0.89       129

    accuracy                           0.89       250
   macro avg       0.89      0.89      0.89       250
weighted avg       0.89      0.89      0.89       250


Площадь под кривой: 0.8894227689153693
```

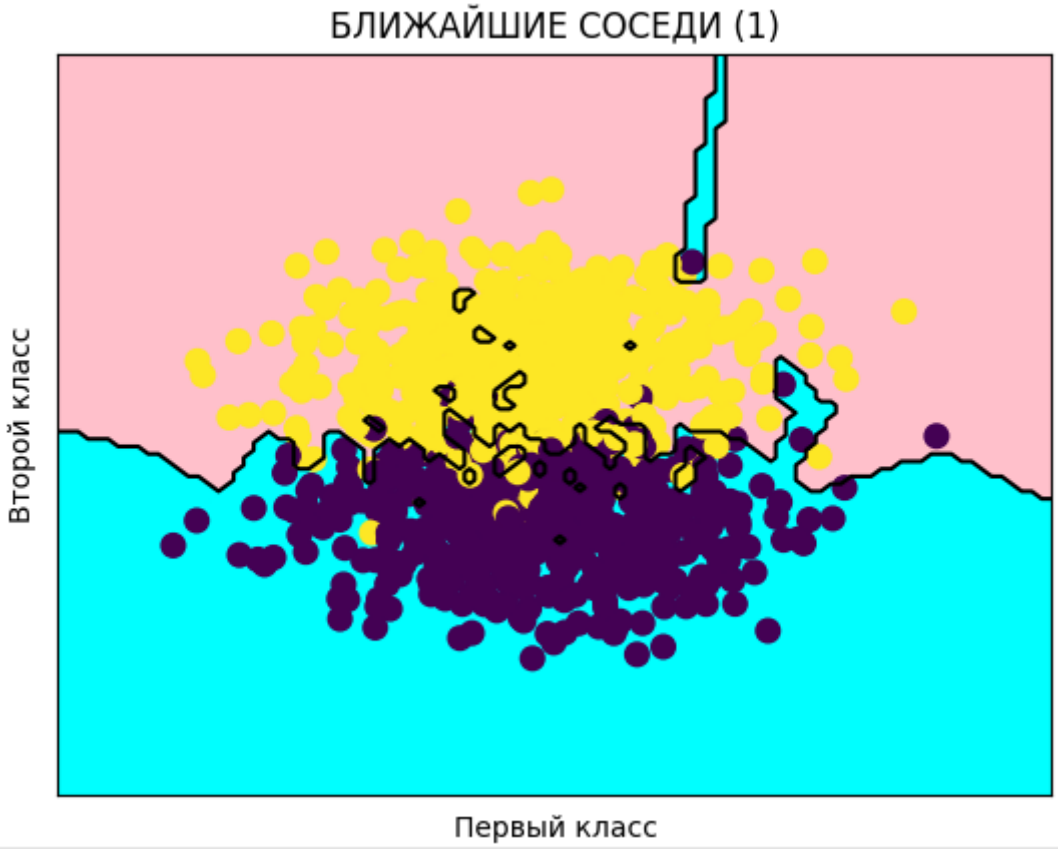

Рисунок 7 – Результат классификации с помощью метода к-ближайших соседей с параметром n_neighbors = 1

Использование параметра n_neighbors = 3. Составленный код для использования данного классификатора с данным параметром представлен на рисунке 8. Полученная информация о точности классификации при использовании данного метода представлена на рисунке 9.

```
knn = KNeighborsClassifier(n_neighbors=3, metric='euclidean')

# Обучаем модель данных
knn.fit(X_train, y_train)

# Оцениваем качество модели
prediction = knn.predict(X_test)

# Выводим сводную информацию
show_info(knn, 'ближайшие соседи (3)', y_test, prediction)
```

Рисунок 8 – Код для классификатора с помощью метода к-ближайших соседей с параметром n_neighbors = 3

```
Матрица неточностей
[[112   9]
 [ 13 116]]

Точность классификации: 0.912

Полнота:
              precision    recall  f1-score   support

           0       0.90      0.93      0.91       121
           1       0.93      0.90      0.91       129

    accuracy                           0.91       250
   macro avg       0.91      0.91      0.91       250
weighted avg       0.91      0.91      0.91       250


Площадь под кривой: 0.9124223204561471
```

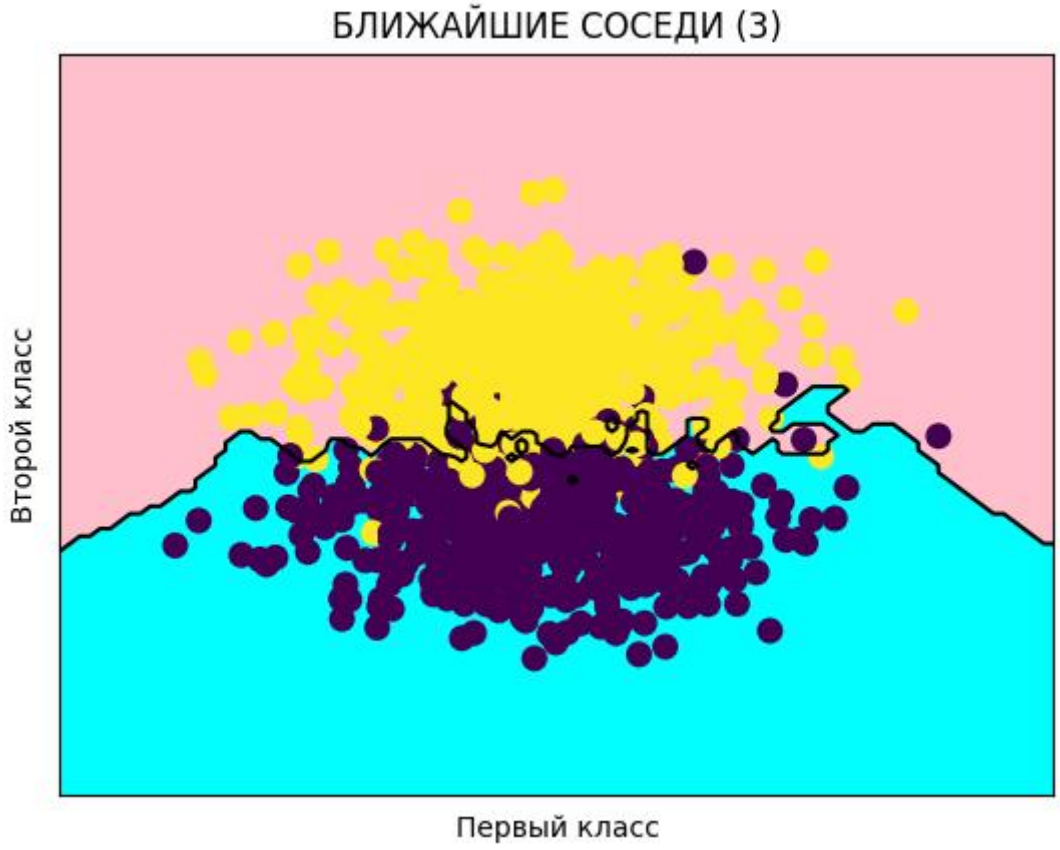

Рисунок 9 – Результат классификации с помощью метода к-ближайших соседей с параметром n_neighbors = 3

Использование параметра n_neighbors = 5. Составленный код для использования данного классификатора с данным параметром представлен на рисунке 10. Полученная информация о точности классификации при использовании данного метода представлена на рисунке 11.

```
knn = KNeighborsClassifier(n_neighbors=5, metric='euclidean')

# Обучаем модель данных
knn.fit(X_train, y_train)

# Оцениваем качество модели
prediction = knn.predict(X_test)

# Выводим сводную информацию
show_info(knn, 'ближайшие соседи (5)', y_test, prediction)
```

Рисунок 10 – Код для классификатора с помощью метода к-ближайших соседей с параметром n_neighbors = 5

```
Матрица неточностей
[[112   9]
 [ 16 113]]

Точность классификации: 0.9

Полнота:
              precision    recall  f1-score   support

           0       0.88      0.93      0.90       121
           1       0.93      0.88      0.90       129

    accuracy                           0.90       250
   macro avg       0.90      0.90      0.90       250
weighted avg       0.90      0.90      0.90       250


Площадь под кривой: 0.9007944134794029
```

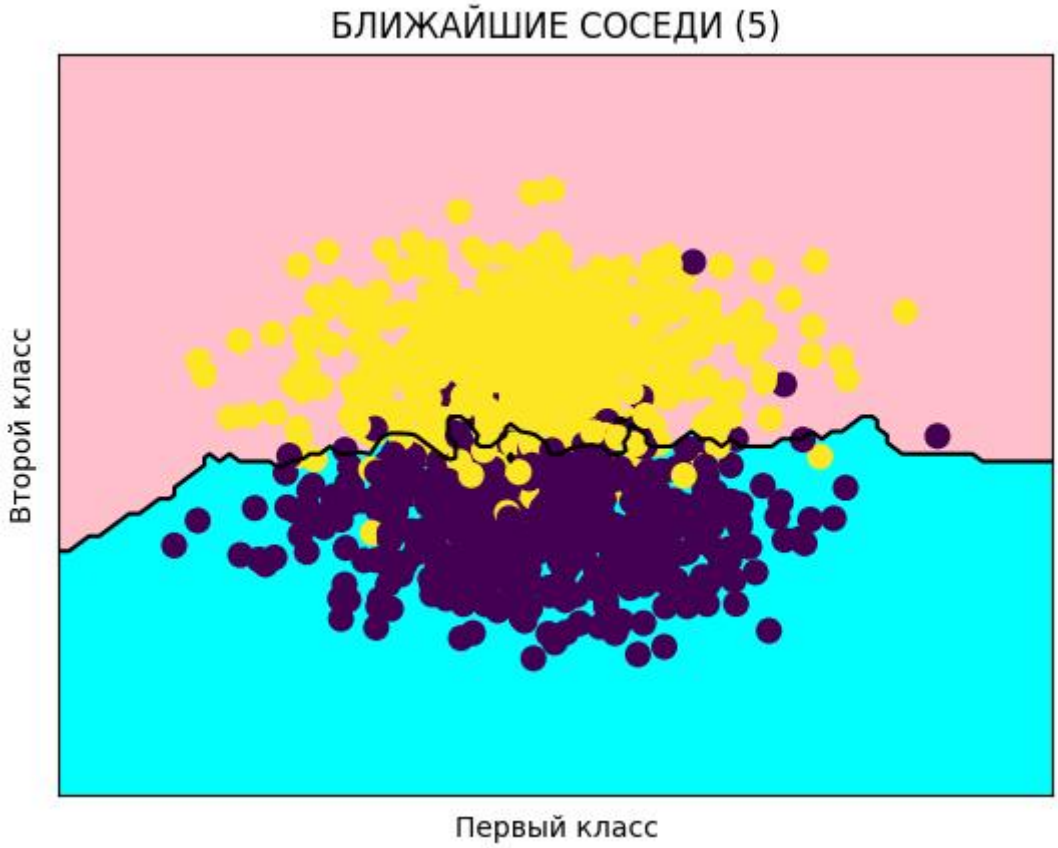

Рисунок 11 – Результат классификации с помощью метода к-ближайших соседей с параметром n_neighbors = 5

Использование параметра n_neighbors = 9. Составленный код для использования данного классификатора с данным параметром представлен на рисунке 12. Полученная информация о точности классификации при использовании данного метода представлена на рисунке 13.

```
knn = KNeighborsClassifier(n_neighbors=9, metric='euclidean')

# Обучаем модель данных
knn.fit(X_train, y_train)

# Оцениваем качество модели
prediction = knn.predict(X_test)

# Выводим сводную информацию
show_info(knn, 'ближайшие соседи (9)', y_test, prediction)
```

Рисунок 12 – Код для классификатора с помощью метода к-ближайших соседей с параметром n_neighbors = 9

```
Матрица неточностей
[[116   5]
 [ 15 114]]

Точность классификации: 0.92

Полнота:
              precision    recall  f1-score   support

           0       0.89      0.96      0.92       121
           1       0.96      0.88      0.92       129

    accuracy                           0.92       250
   macro avg       0.92      0.92      0.92       250
weighted avg       0.92      0.92      0.92       250


Площадь под кривой: 0.9211993080914858
```

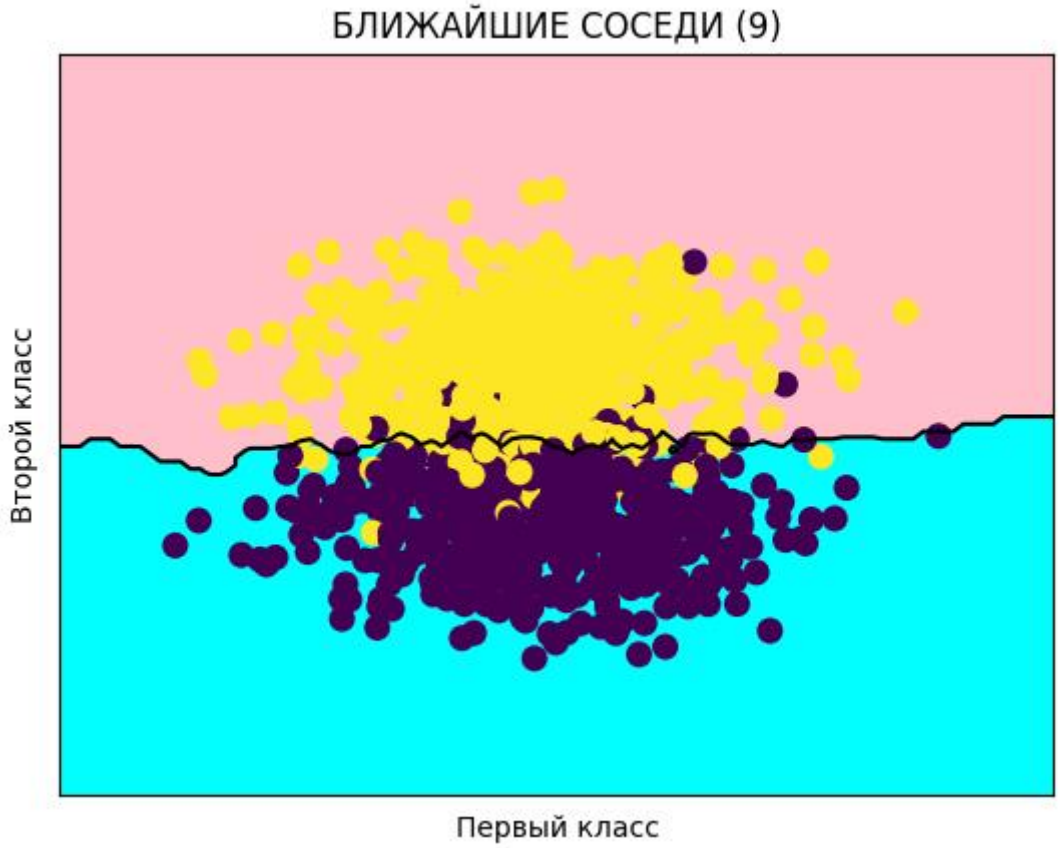

Рисунок 13 – Результат классификации с помощью метода к-ближайших соседей с параметром n_neighbors = 9

Классификация с помощью наивного байесовского классификатора

Составленный код для использования данного классификатора представлен на рисунке 14. Полученная информация о точности классификации при использовании данного метода представлена на рисунке 15.

```
from sklearn.naive_bayes import GaussianNB

nb = GaussianNB()

# Обучаем модель данных
nb.fit(X_train, y_train)

# Оцениваем качество модели
prediction = nb.predict(X_test)

# Выводим сводную информацию
show_info(nb, 'Наивный байесовский классификатор', y_test, prediction)
```

Рисунок 14 – Код для классификатора с помощью наивного байесовского классификатора

```
Матрица неточностей
[[117   4]
 [ 20 109]]

Точность классификации: 0.904

Полнота:
              precision    recall  f1-score   support

           0       0.85      0.97      0.91       121
           1       0.96      0.84      0.90       129

    accuracy                           0.90       250
   macro avg       0.91      0.91      0.90       250
weighted avg       0.91      0.90      0.90       250


Площадь под кривой: 0.9059516945352041
```

Рисунок 15 – Результат классификации с помощью наивного байесовского классификатора

Классификация с помощью случайного леса

Использование параметра n_estimators = 5. Составленный код для использования данного классификатора с данным параметром представлен на рисунке 16. Полученная информация о точности классификации при использовании данного метода представлена на рисунке 17.

```
from sklearn.ensemble import RandomForestClassifier

rfc = RandomForestClassifier(n_estimators=5)

# Обучаем модель данных
rfc.fit(X_train, y_train)

# Оцениваем качество модели
prediction = rfc.predict(X_test)

# Выводим сводную информацию
show_info(rfc, 'случайный лес (5)', y_test, prediction)
```

Рисунок 16 – Код для классификатора с помощью случайного леса с параметром n_ estimators = 5

```
Матрица неточностей
[[114   7]
 [ 16 113]]

Точность классификации: 0.908

Полнота:
              precision    recall  f1-score   support

           0       0.88      0.94      0.91       121
           1       0.94      0.88      0.91       129

    accuracy                           0.91       250
   macro avg       0.91      0.91      0.91       250
weighted avg       0.91      0.91      0.91       250


Площадь под кривой: 0.9090588762893203
```

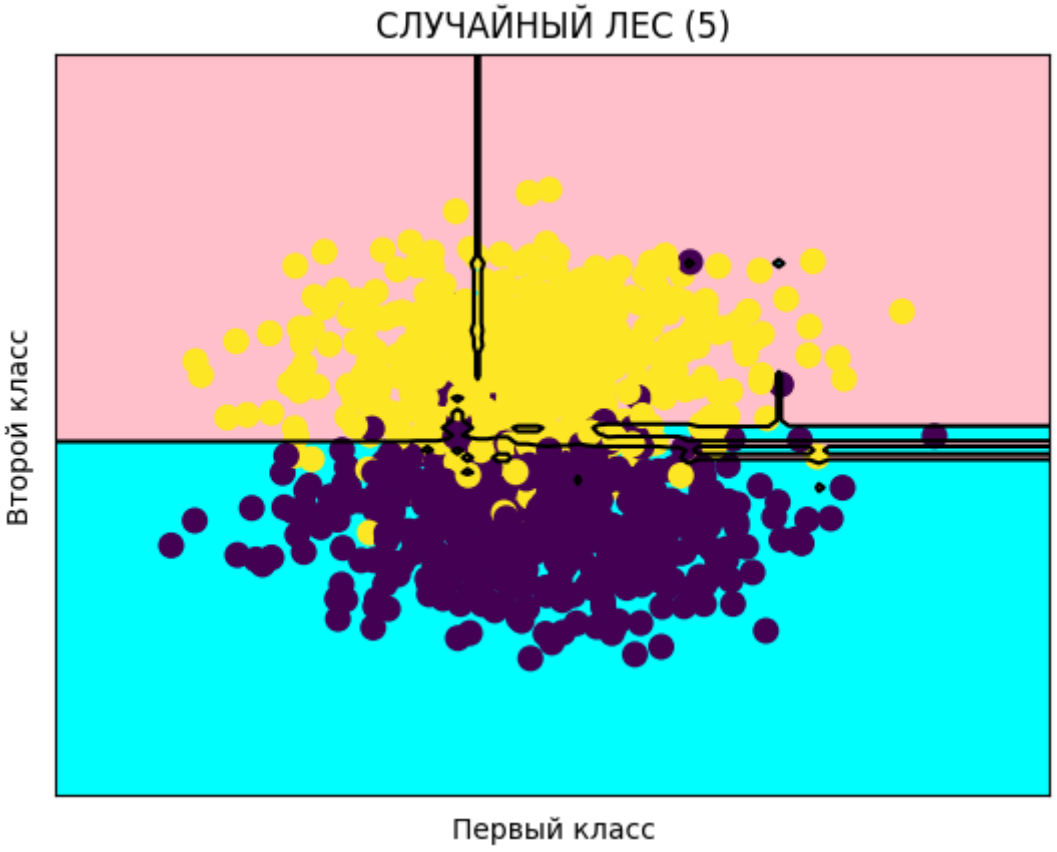

Рисунок 17 – Результат классификации с помощью случайного леса с параметром n_ estimators = 5

Использование параметра n_estimators = 10. Составленный код для использования данного классификатора с данным параметром представлен на рисунке 18. Полученная информация о точности классификации при использовании данного метода представлена на рисунке 19.

```
rfc = RandomForestClassifier(n_estimators=10)

# Обучаем модель данных
rfc.fit(X_train, y_train)

# Оцениваем качество модели
prediction = rfc.predict(X_test)

# Выводим сводную информацию
show_info(rfc, 'случайный лес (10)', y_test, prediction)
```

Рисунок 18 – Код для классификатора с помощью случайного леса с параметром n_ estimators = 10

```
Матрица неточностей
[[116   5]
 [ 16 113]]

Точность классификации: 0.916

Полнота:
              precision    recall  f1-score   support

           0       0.88      0.96      0.92       121
           1       0.96      0.88      0.91       129

    accuracy                           0.92       250
   macro avg       0.92      0.92      0.92       250
weighted avg       0.92      0.92      0.92       250


Площадь под кривой: 0.9173233390992377
```

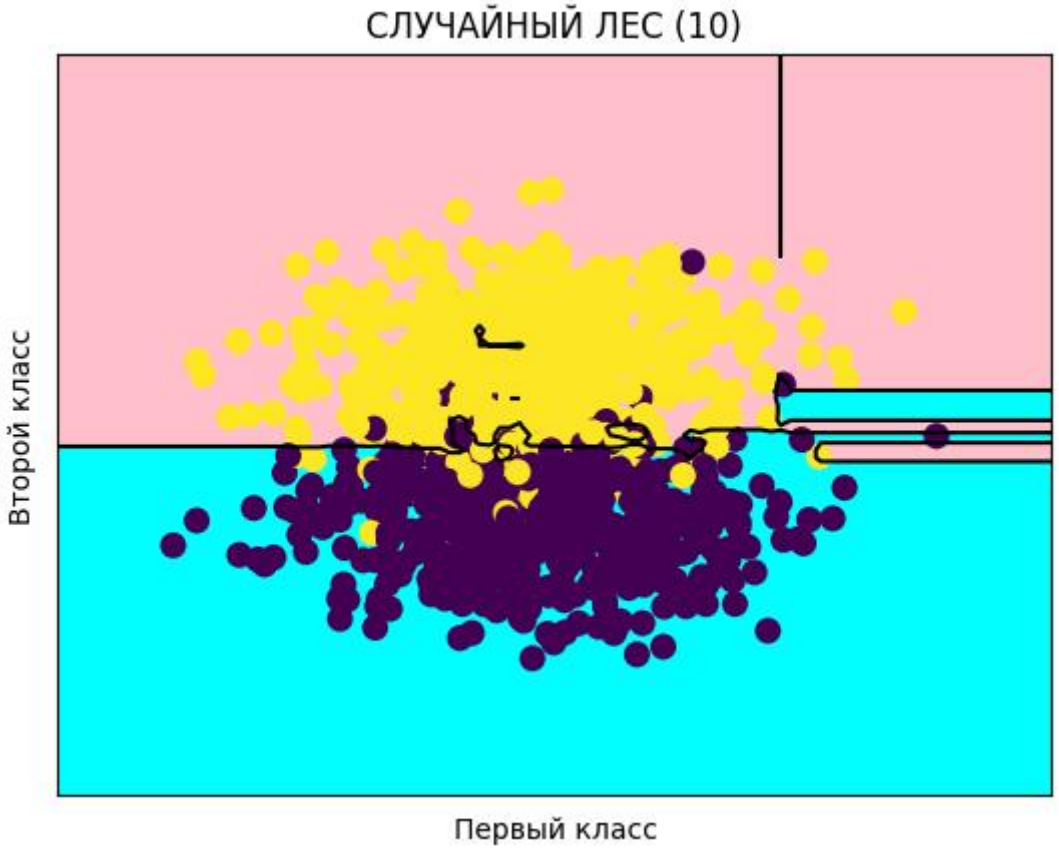

Рисунок 19 – Результат классификации с помощью случайного леса с параметром n_ estimators = 10

Использование параметра n_estimators = 15. Составленный код для использования данного классификатора с данным параметром представлен на рисунке 20. Полученная информация о точности классификации при использовании данного метода представлена на рисунке 21.

```
rfc = RandomForestClassifier(n_estimators=15)

# Обучаем модель данных
rfc.fit(X_train, y_train)

# Оцениваем качество модели
prediction = rfc.predict(X_test)

# Выводим сводную информацию
show_info(rfc, 'случайный лес (15)', y_test, prediction)
```

Рисунок 20 – Код для классификатора с помощью случайного леса с параметром n_ estimators = 15

```
Матрица неточностей
[[114   7]
 [ 14 115]]

Точность классификации: 0.916

Полнота:
              precision    recall  f1-score   support

           0       0.89      0.94      0.92       121
           1       0.94      0.89      0.92       129

    accuracy                           0.92       250
   macro avg       0.92      0.92      0.92       250
weighted avg       0.92      0.92      0.92       250


Площадь под кривой: 0.9168108142738164
```

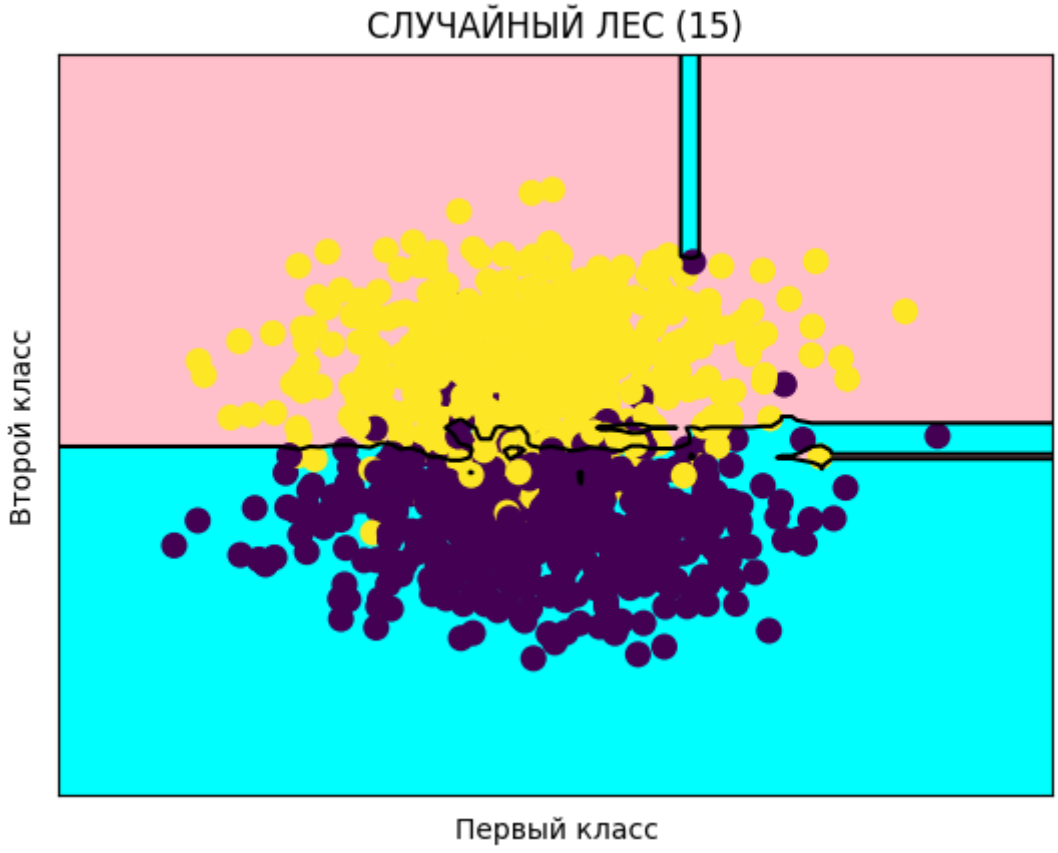

Рисунок 21 – Результат классификации с помощью случайного леса с параметром n_ estimators = 15

Использование параметра n_estimators = 20. Составленный код для использования данного классификатора с данным параметром представлен на рисунке 22. Полученная информация о точности классификации при использовании данного метода представлена на рисунке 23.

```
rfc = RandomForestClassifier(n_estimators=20)

# Обучаем модель данных
rfc.fit(X_train, y_train)

# Оцениваем качество модели
prediction = rfc.predict(X_test)

# Выводим сводную информацию
show_info(rfc, 'случайный лес (20)', y_test, prediction)
```

Рисунок 22 – Код для классификатора с помощью случайного леса с параметром n_ estimators = 20

```
Матрица неточностей
[[115   6]
 [ 15 114]]

Точность классификации: 0.916

Полнота:
              precision    recall  f1-score   support

           0       0.88      0.95      0.92       121
           1       0.95      0.88      0.92       129

    accuracy                           0.92       250
   macro avg       0.92      0.92      0.92       250
weighted avg       0.92      0.92      0.92       250


Площадь под кривой: 0.9170670766865269
```

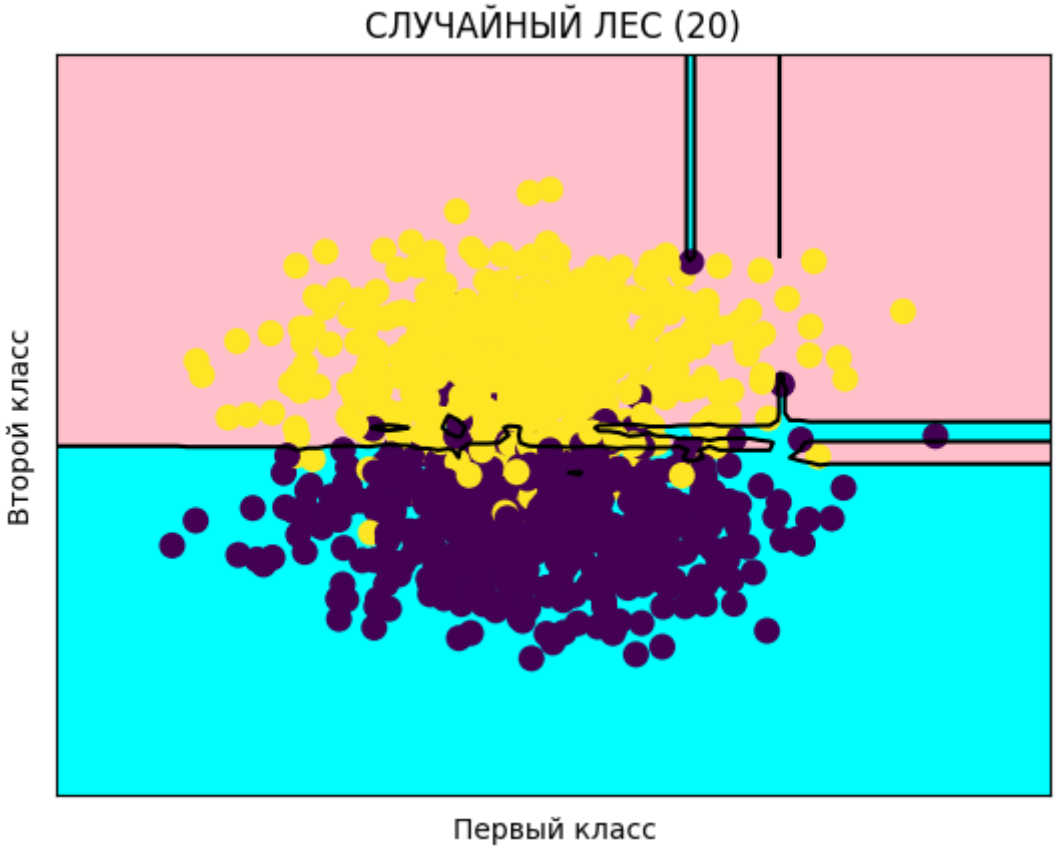

Рисунок 23 – Результат классификации с помощью случайного леса с параметром n_ estimators = 20

Использование параметра n_estimators = 50. Составленный код для использования данного классификатора с данным параметром представлен на рисунке 24. Полученная информация о точности классификации при использовании данного метода представлена на рисунке 25.

```
rfc = RandomForestClassifier(n_estimators=50)

# Обучаем модель данных
rfc.fit(X_train, y_train)

# Оцениваем качество модели
prediction = rfc.predict(X_test)

# Выводим сводную информацию
show_info(rfc, 'случайный лес (50)', y_test, prediction)
```

Рисунок 24 – Код для классификатора с помощью случайного леса с параметром n_ estimators = 50

```
Матрица неточностей
[[116   5]
 [ 16 113]]

Точность классификации: 0.916

Полнота:
              precision    recall  f1-score   support

           0       0.88      0.96      0.92       121
           1       0.96      0.88      0.91       129

    accuracy                           0.92       250
   macro avg       0.92      0.92      0.92       250
weighted avg       0.92      0.92      0.92       250


Площадь под кривой: 0.9173233390992377
```

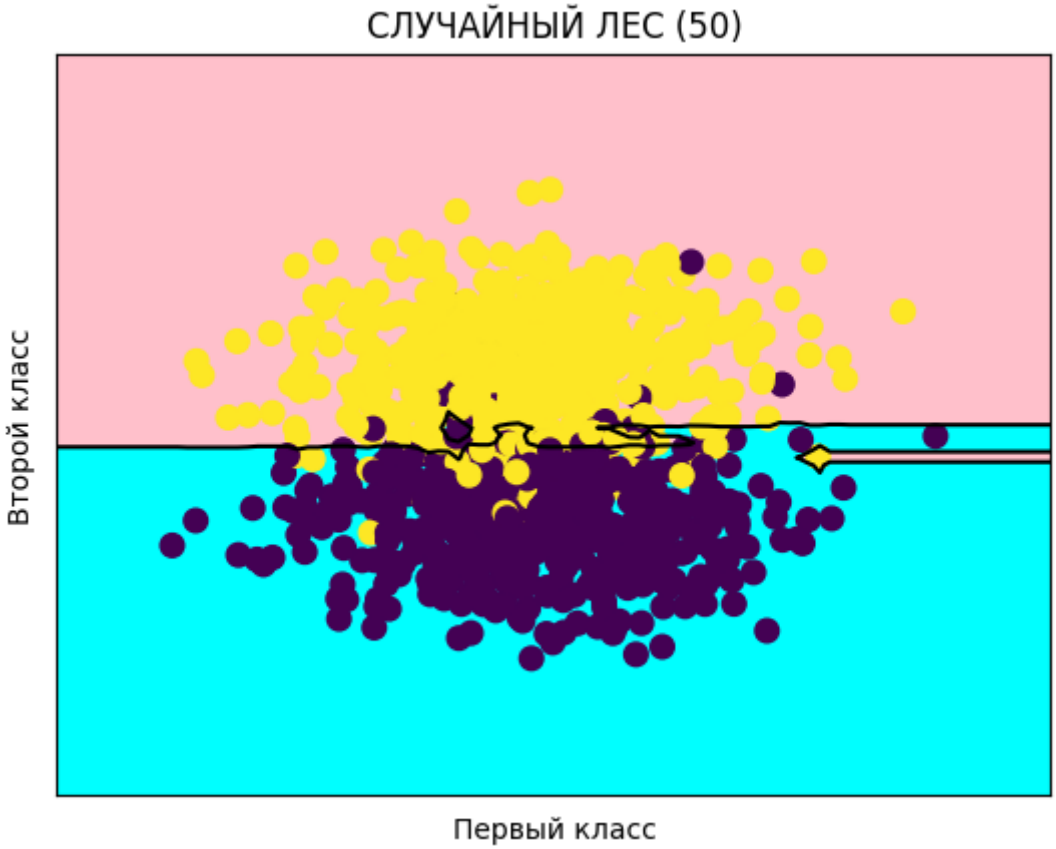

Рисунок 24 – Результат классификации с помощью случайного леса с параметром n_ estimators = 50

Анализ результатов

Таблица 1 – Результат классификации по методам при 75% обучающей выборки

Метод с параметрами	Точность	Площадь под кривой
Метод к-ближайших соседей (n_neighbors = 1)	0.888	0.889
Метод к-ближайших соседей (n_neighbors = 3)	0.912	0.912
Метод к-ближайших соседей (n_neighbors = 5)	0.9	0.9
Метод к-ближайших соседей (n_neighbors = 9)	0.92	0.921
Наивный байесовский классификатор	0.904	0.906
Случайный лес (n_estimators = 5)	0.908	0.909
Случайный лес (n_estimators = 10)	0.916	0.917
Случайный лес (n_estimators = 15)	0.916	0.917
Случайный лес (n_estimators = 20)	0.916	0.917
Случайный лес (n_estimators = 50)	0.916	0.917

По результатам представленных в таблице 1 можно сделать вывод, что лучше всего себя показали метод к-ближайших соседей при значении параметра n_neighbors = 9 и метод случайного леса при значении параметра n_estimators = 10 и выше.

Рассмотрим случай уменьшения тестовой выборки. Установим, что тестовая выборка составляет 10% и построим графики визуализации обучающей и тестовой выборки, данные графики представлены на рисунках 25 и 26 соответственно.

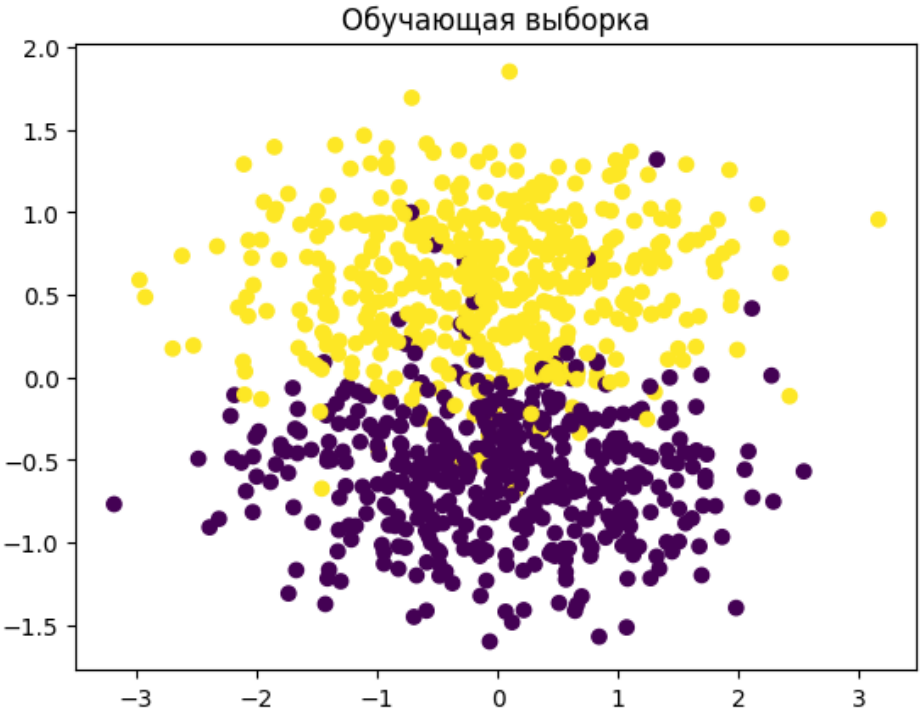

Рисунок 25 – Обучающая выборка при 90% от общего размера

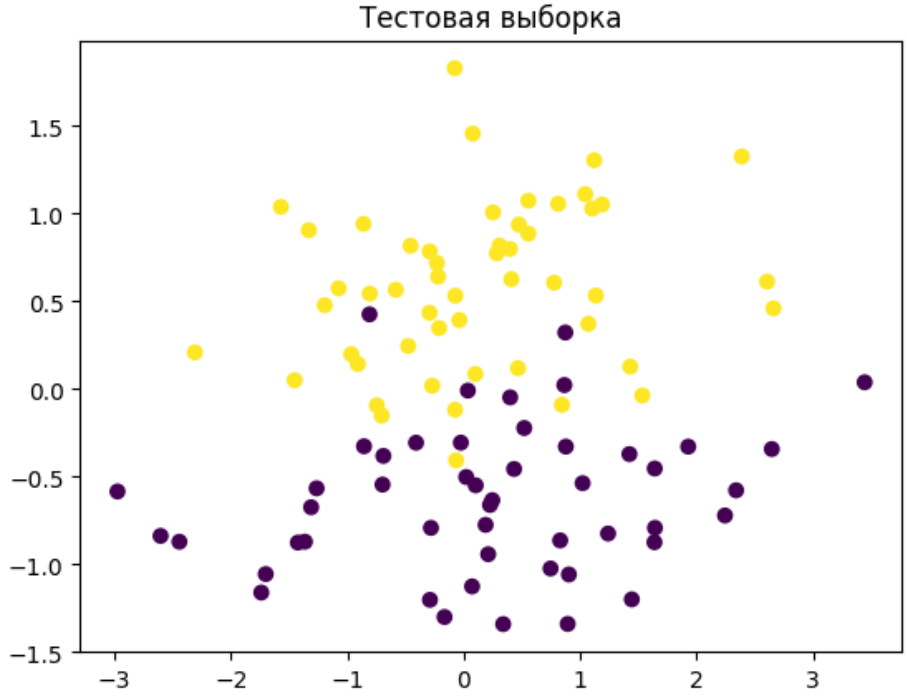

Рисунок 26 – Тестовая выборка при 10% от общего размера

Таблица 2 – Результат классификации по методам при 90% обучающей выборки

Метод с параметрами	Точность	Площадь под кривой
Метод к-ближайших соседей (n_neighbors = 1)	0.92	0.92
Метод к-ближайших соседей (n_neighbors = 3)	0.91	0.91
Метод к-ближайших соседей (n_neighbors = 5)	0.9	0.9
Метод к-ближайших соседей (n_neighbors = 9)	0.9	0.9
Наивный байесовский классификатор	0.9	0.9
Случайный лес (n_estimators = 5)	0.88	0.88
Случайный лес (n_estimators = 10)	0.9	0.9
Случайный лес (n_estimators = 15)	0.88	0.88
Случайный лес (n_estimators = 20)	0.91	0.91
Случайный лес (n_estimators = 50)	0.91	0.91

По результатам представленных в таблице 2 можно сделать вывод, что лучше всего себя показали метод к-ближайших соседей при значении параметра n_neighbors = 1 (его точность составила 92%) и метод случайного леса при значении параметра n_estimators = 20 и выше (его точность составила 91%).

Рассмотрим случай уменьшения тестовой выборки. Установим, что тестовая выборка составляет 35% и построим графики визуализации обучающей и тестовой выборки, данные графики представлены на рисунках 27 и 28 соответственно.

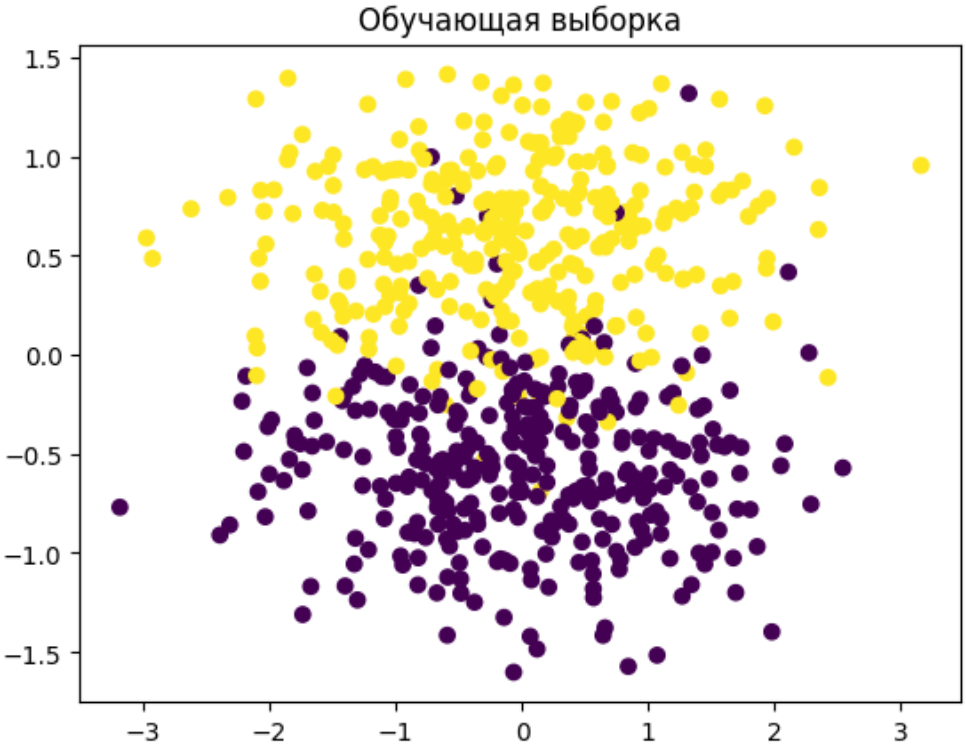

Рисунок 27 – Обучающая выборка при 65% от общего размера

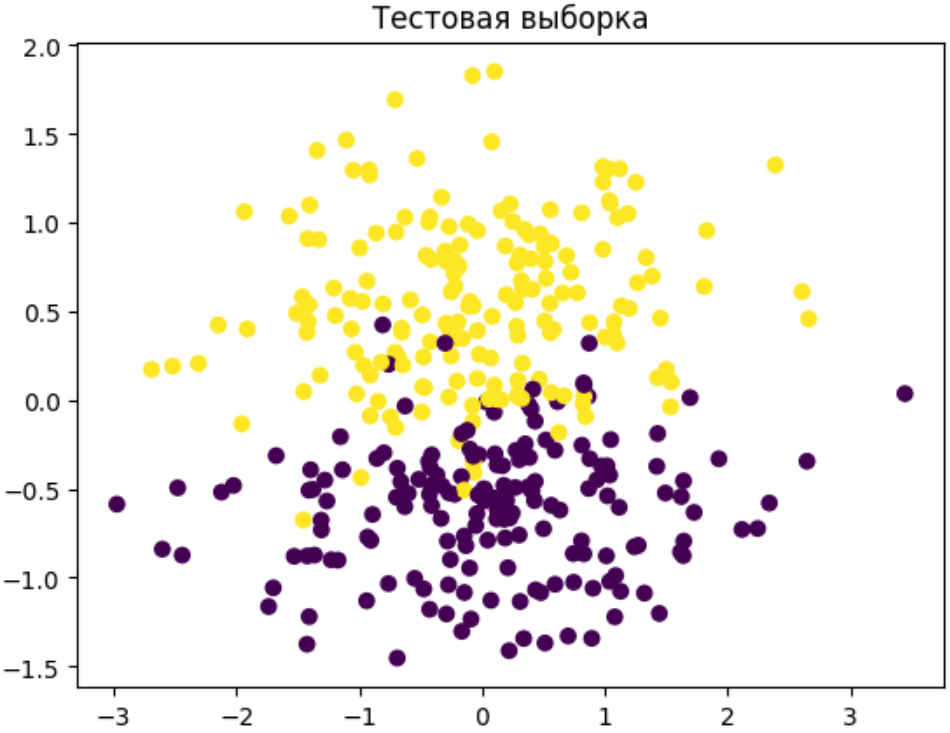

Рисунок 28 – Тестовая выборка при 35% от общего размера

Таблица 3 – Результат классификации по методам при 65% обучающей выборки

Метод с параметрами	Точность	Площадь под кривой
Метод к-ближайших соседей (n_neighbors = 1)	0.87	0.87
Метод к-ближайших соседей (n_neighbors = 3)	0.91	0.91
Метод к-ближайших соседей (n_neighbors = 5)	0.91	0.91
Метод к-ближайших соседей (n_neighbors = 9)	0.91	0.91
Наивный байесовский классификатор	0.9	0.9
Случайный лес (n_estimators = 5)	0.91	0.91
Случайный лес (n_estimators = 10)	0.9	0.9
Случайный лес (n_estimators = 15)	0.91	0.91
Случайный лес (n_estimators = 20)	0.91	0.91
Случайный лес (n_estimators = 50)	0.91	0.91

По результатам представленных в таблице 3 можно сделать вывод, что лучше всего себя показали метод к-ближайших соседей (его точность составила 91%) и метод случайного леса (его точность составила 91%).

Заключение

В ходе выполнения данной лабораторной работы мною были изучены методы классификации на примере бинарной классификации фактографических данных. Для классификации использовались метод к-ближайших соседей, наивный байесовский классификатор и случайный лес. Для каждого из классификатора был проведен анализ в ходе которого изменялись параметры классификации и наблюдение за их влиянием на точность классификации. Из анализа моделей классификаторов были составлены таблицы, в которых сравниваются различные методы и их метрики. Также были рассмотрены влияние объема выборки на точность – для хорошей точности необходимо подбирать не слишком маленькую долю для тестовой выборки.

Приложение А

Исходный код при 25% тестовой выборки

```
#!/usr/bin/env python
# coding: utf-8

# # Задание
# ---
# ## Для всех:
# ### `n_features = 2`
# ### `n_redundant = 0`
# ### `n_informative = 1`
# ### `n_clusters_per_class = 1`
# ---
# ## Для варианта №11
# ### Вид классов: `classification`
# ### Random state: `15`
# ### Class sep: `0.6`

# In[1]:


import numpy as np

from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split


# In[2]:


import matplotlib.pyplot as plt


# In[3]:


def plot_2d_separator(classifier, X, fill=False, line=True, ax=None,
eps=None):
    if eps is None:
        eps = 1.0
    x_min, x_max = X[:, 0].min() - eps, X[:, 0].max() + eps
    y_min, y_max = X[:, 1].min() - eps, X[:, 1].max() + eps
    xx = np.linspace(x_min, x_max, 100)
    yy = np.linspace(y_min, y_max, 100)
    x1, x2 = np.meshgrid(xx, yy)
    X_grid = np.c_[x1.ravel(), x2.ravel()]
    try:
        decision_values = classifier.decision_function(X_grid)
        levels = [0]
        fill_levels = [decision_values.min(), 0, decision_values.max()]
    except AttributeError:
        decision_values = classifier.predict_proba(X_grid)[:, 1]
        levels = [.5]
        fill_levels = [0, .5, 1]

    if ax is None:
        ax = plt.gca()
    if fill:
        ax.contourf(x1,
                    x2,
                    decision_values.reshape(x1.shape),
```

```
                     levels=fill_levels,
                     colors=['cyan', 'pink', 'yellow'])
    if line:
        ax.contour(x1,
                   x2,
                   decision_values.reshape(x1.shape),
                   levels=levels,
                   colors='black')

    ax.set_xlim(x_min, x_max)
    ax.set_ylim(y_min, y_max)
    ax.set_xticks(())
    ax.set_yticks(())


# ## Генерация выборки

# In[4]:


X, y = make_classification(n_features=2,
                           n_samples=1000,
                           n_redundant=0,
                           n_informative=1,
                           n_clusters_per_class=1,
                           random_state=15,
                           class_sep=0.6)


# In[5]:


print('Координаты точек: ')
print(X[:15])
print('Метки класса: ')
print(y[:15])


#

# In[6]:


plt.scatter(X[:, 0], X[:, 1], c=y)
plt.show()


# ## Разбитие выборки на обучающее и тестовое множество

# In[7]:


X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    test_size=0.25,
                                                    random_state=42)


# In[8]:


plt.title('Обучающая выборка')
plt.scatter(X_train[:, 0], X_train[:, 1], c=y_train)
```

```
plt.show()


# In[9]:


plt.title('Тестовая выборка')
plt.scatter(X_test[:, 0], X_test[:, 1], c=y_test)
plt.show()


# ## Кластеризация

# In[10]:


from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.metrics import roc_auc_score

def show_info(classifier, classifier_name, real_values, prediction_values):
    print(f'Метод классификации: {classifier_name}\n')

    # Выводим предсказанное и реальное значение
    print('Предсказанные и реальные значения:')
    print(prediction_values)
    print(real_values)

    # Выводим матрицу неточностей
    print('\nМатрица неточностей')
    print(confusion_matrix(real_values, prediction_values))

    # Выводим точность классификации
    print(f'\nТочность классификации: {accuracy_score(prediction_values,
real_values)}')

    # Выводим полноту
    print('\nПолнота: ')
    print(classification_report(real_values, prediction_values))

    # AUC ROC
    print(f'\nПлощадь под кривой: {roc_auc_score(real_values,
prediction_values)}')

    plt.xlabel('Первый класс')
    plt.ylabel('Второй класс')
    plt.title(classifier_name.upper())
    plot_2d_separator(classifier, X, fill=True)
    plt.scatter(X[:, 0], X[:, 1], c=y, s=70)


# ### Метод k-ближайших соседей (1)

# In[11]:


from sklearn.neighbors import KNeighborsClassifier


# In[12]:
```

```
knn = KNeighborsClassifier(n_neighbors=1, metric='euclidean')

# Обучаем модель данных
knn.fit(X_train, y_train)

# Оцениваем качество модели
prediction = knn.predict(X_test)

# Выводим сводную информацию
show_info(knn, 'ближайшие соседи (1)', y_test, prediction)


# In[13]:


knn = KNeighborsClassifier(n_neighbors=3, metric='euclidean')

# Обучаем модель данных
knn.fit(X_train, y_train)

# Оцениваем качество модели
prediction = knn.predict(X_test)

# Выводим сводную информацию
show_info(knn, 'ближайшие соседи (3)', y_test, prediction)


# In[14]:


knn = KNeighborsClassifier(n_neighbors=5, metric='euclidean')

# Обучаем модель данных
knn.fit(X_train, y_train)

# Оцениваем качество модели
prediction = knn.predict(X_test)

# Выводим сводную информацию
show_info(knn, 'ближайшие соседи (5)', y_test, prediction)


# In[15]:


knn = KNeighborsClassifier(n_neighbors=9, metric='euclidean')

# Обучаем модель данных
knn.fit(X_train, y_train)

# Оцениваем качество модели
prediction = knn.predict(X_test)

# Выводим сводную информацию
show_info(knn, 'ближайшие соседи (9)', y_test, prediction)


# ## Наивный байесовский классификатор

# In[22]:


from sklearn.naive_bayes import GaussianNB
```

```
nb = GaussianNB()

# Обучаем модель данных
nb.fit(X_train, y_train)

# Оцениваем качество модели
prediction = nb.predict(X_test)

# Выводим сводную информацию
show_info(nb, 'Наивный байесовский классификатор', y_test, prediction)


# ## Случайный лес

# In[17]:


from sklearn.ensemble import RandomForestClassifier

rfc = RandomForestClassifier(n_estimators=5)

# Обучаем модель данных
rfc.fit(X_train, y_train)

# Оцениваем качество модели
prediction = rfc.predict(X_test)

# Выводим сводную информацию
show_info(rfc, 'случайный лес (5)', y_test, prediction)


# In[18]:


rfc = RandomForestClassifier(n_estimators=10)

# Обучаем модель данных
rfc.fit(X_train, y_train)

# Оцениваем качество модели
prediction = rfc.predict(X_test)

# Выводим сводную информацию
show_info(rfc, 'случайный лес (10)', y_test, prediction)


# In[19]:


rfc = RandomForestClassifier(n_estimators=15)

# Обучаем модель данных
rfc.fit(X_train, y_train)

# Оцениваем качество модели
prediction = rfc.predict(X_test)

# Выводим сводную информацию
show_info(rfc, 'случайный лес (15)', y_test, prediction)


# In[20]:
```

```
rfc = RandomForestClassifier(n_estimators=20)

# Обучаем модель данных
rfc.fit(X_train, y_train)

# Оцениваем качество модели
prediction = rfc.predict(X_test)

# Выводим сводную информацию
show_info(rfc, 'случайный лес (20)', y_test, prediction)


# In[21]:


rfc = RandomForestClassifier(n_estimators=50)

# Обучаем модель данных
rfc.fit(X_train, y_train)

# Оцениваем качество модели
prediction = rfc.predict(X_test)

# Выводим сводную информацию
show_info(rfc, 'случайный лес (50)', y_test, prediction)
```

Приложение Б

Исходный код при 10% тестовой выборки

```
#!/usr/bin/env python
# coding: utf-8

# # Задание
# ---
# ## Для всех:
# ### `n_features = 2`
# ### `n_redundant = 0`
# ### `n_informative = 1`
# ### `n_clusters_per_class = 1`
# ---
# ## Для варианта №11
# ### Вид классов: `classification`
# ### Random state: `15`
# ### Class sep: `0.6`

# In[1]:


import numpy as np

from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split


# In[2]:


import matplotlib.pyplot as plt


# In[3]:


def plot_2d_separator(classifier, X, fill=False, line=True, ax=None,
eps=None):
    if eps is None:
        eps = 1.0
    x_min, x_max = X[:, 0].min() - eps, X[:, 0].max() + eps
    y_min, y_max = X[:, 1].min() - eps, X[:, 1].max() + eps
    xx = np.linspace(x_min, x_max, 100)
    yy = np.linspace(y_min, y_max, 100)
    x1, x2 = np.meshgrid(xx, yy)
    X_grid = np.c_[x1.ravel(), x2.ravel()]
    try:
        decision_values = classifier.decision_function(X_grid)
        levels = [0]
        fill_levels = [decision_values.min(), 0, decision_values.max()]
    except AttributeError:
        decision_values = classifier.predict_proba(X_grid)[:, 1]
        levels = [.5]
        fill_levels = [0, .5, 1]

    if ax is None:
        ax = plt.gca()
    if fill:
        ax.contourf(x1,
                    x2,
                    decision_values.reshape(x1.shape),
```

```
                    levels=fill_levels,
                    colors=['cyan', 'pink', 'yellow'])
    if line:
        ax.contour(x1,
                   x2,
                   decision_values.reshape(x1.shape),
                   levels=levels,
                   colors='black')

    ax.set_xlim(x_min, x_max)
    ax.set_ylim(y_min, y_max)
    ax.set_xticks(())
    ax.set_yticks(())


# ## Генерация выборки

# In[4]:


X, y = make_classification(n_features=2,
                           n_samples=1000,
                           n_redundant=0,
                           n_informative=1,
                           n_clusters_per_class=1,
                           random_state=15,
                           class_sep=0.6)


# In[5]:


print('Координаты точек: ')
print(X[:15])
print('Метки класса: ')
print(y[:15])


#

# In[6]:


plt.scatter(X[:, 0], X[:, 1], c=y)
plt.show()


# ## Разбитие выборки на обучающее и тестовое множество

# In[7]:


X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    test_size=0.10,
                                                    random_state=42)


# In[8]:


plt.title('Обучающая выборка')
plt.scatter(X_train[:, 0], X_train[:, 1], c=y_train)
```

```
plt.show()


# In[9]:


plt.title('Тестовая выборка')
plt.scatter(X_test[:, 0], X_test[:, 1], c=y_test)
plt.show()


# ## Кластеризация

# In[10]:


from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.metrics import roc_auc_score

def show_info(classifier, classifier_name, real_values, prediction_values):
    print(f'Метод классификации: {classifier_name}\n')

    # Выводим предсказанное и реальное значение
    print('Предсказанные и реальные значения:')
    print(prediction_values)
    print(real_values)

    # Выводим матрицу неточностей
    print('\nМатрица неточностей')
    print(confusion_matrix(real_values, prediction_values))

    # Выводим точность классификации
    print(f'\nТочность классификации: {accuracy_score(prediction_values,
real_values)}')

    # Выводим полноту
    print('\nПолнота: ')
    print(classification_report(real_values, prediction_values))

    # AUC ROC
    print(f'\nПлощадь под кривой: {roc_auc_score(real_values,
prediction_values)}')

    plt.xlabel('Первый класс')
    plt.ylabel('Второй класс')
    plt.title(classifier_name.upper())
    plot_2d_separator(classifier, X, fill=True)
    plt.scatter(X[:, 0], X[:, 1], c=y, s=70)


# ### Метод k-ближайших соседей (1)

# In[11]:


from sklearn.neighbors import KNeighborsClassifier


# In[12]:
```

```
knn = KNeighborsClassifier(n_neighbors=1, metric='euclidean')

# Обучаем модель данных
knn.fit(X_train, y_train)

# Оцениваем качество модели
prediction = knn.predict(X_test)

# Выводим сводную информацию
show_info(knn, 'ближайшие соседи (1)', y_test, prediction)


# In[13]:


knn = KNeighborsClassifier(n_neighbors=3, metric='euclidean')

# Обучаем модель данных
knn.fit(X_train, y_train)

# Оцениваем качество модели
prediction = knn.predict(X_test)

# Выводим сводную информацию
show_info(knn, 'ближайшие соседи (3)', y_test, prediction)


# In[14]:


knn = KNeighborsClassifier(n_neighbors=5, metric='euclidean')

# Обучаем модель данных
knn.fit(X_train, y_train)

# Оцениваем качество модели
prediction = knn.predict(X_test)

# Выводим сводную информацию
show_info(knn, 'ближайшие соседи (5)', y_test, prediction)


# In[15]:


knn = KNeighborsClassifier(n_neighbors=9, metric='euclidean')

# Обучаем модель данных
knn.fit(X_train, y_train)

# Оцениваем качество модели
prediction = knn.predict(X_test)

# Выводим сводную информацию
show_info(knn, 'ближайшие соседи (9)', y_test, prediction)


# ## Наивный байесовский классификатор

# In[16]:


from sklearn.naive_bayes import GaussianNB
```

```
nb = GaussianNB()

# Обучаем модель данных
nb.fit(X_train, y_train)

# Оцениваем качество модели
prediction = nb.predict(X_test)

# Выводим сводную информацию
show_info(nb, 'Наивный байесовский классификатор', y_test, prediction)


# ## Случайный лес

# In[17]:


from sklearn.ensemble import RandomForestClassifier

rfc = RandomForestClassifier(n_estimators=5)

# Обучаем модель данных
rfc.fit(X_train, y_train)

# Оцениваем качество модели
prediction = rfc.predict(X_test)

# Выводим сводную информацию
show_info(rfc, 'случайный лес (5)', y_test, prediction)


# In[18]:


rfc = RandomForestClassifier(n_estimators=10)

# Обучаем модель данных
rfc.fit(X_train, y_train)

# Оцениваем качество модели
prediction = rfc.predict(X_test)

# Выводим сводную информацию
show_info(rfc, 'случайный лес (10)', y_test, prediction)


# In[19]:


rfc = RandomForestClassifier(n_estimators=15)

# Обучаем модель данных
rfc.fit(X_train, y_train)

# Оцениваем качество модели
prediction = rfc.predict(X_test)

# Выводим сводную информацию
show_info(rfc, 'случайный лес (15)', y_test, prediction)


# In[20]:
```

```
rfc = RandomForestClassifier(n_estimators=20)

# Обучаем модель данных
rfc.fit(X_train, y_train)

# Оцениваем качество модели
prediction = rfc.predict(X_test)

# Выводим сводную информацию
show_info(rfc, 'случайный лес (20)', y_test, prediction)


# In[21]:


rfc = RandomForestClassifier(n_estimators=50)

# Обучаем модель данных
rfc.fit(X_train, y_train)

# Оцениваем качество модели
prediction = rfc.predict(X_test)

# Выводим сводную информацию
show_info(rfc, 'случайный лес (50)', y_test, prediction)
```

Приложение В

Исходный код при 35% тестовой выборки

```
#!/usr/bin/env python
# coding: utf-8

# # Задание
# ---
# ## Для всех:
# ### `n_features = 2`
# ### `n_redundant = 0`
# ### `n_informative = 1`
# ### `n_clusters_per_class = 1`
# ---
# ## Для варианта №11
# ### Вид классов: `classification`
# ### Random state: `15`
# ### Class sep: `0.6`

# In[1]:


import numpy as np

from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split


# In[2]:


import matplotlib.pyplot as plt


# In[3]:


def plot_2d_separator(classifier, X, fill=False, line=True, ax=None,
eps=None):
    if eps is None:
        eps = 1.0
    x_min, x_max = X[:, 0].min() - eps, X[:, 0].max() + eps
    y_min, y_max = X[:, 1].min() - eps, X[:, 1].max() + eps
    xx = np.linspace(x_min, x_max, 100)
    yy = np.linspace(y_min, y_max, 100)
    x1, x2 = np.meshgrid(xx, yy)
    X_grid = np.c_[x1.ravel(), x2.ravel()]
    try:
        decision_values = classifier.decision_function(X_grid)
        levels = [0]
        fill_levels = [decision_values.min(), 0, decision_values.max()]
    except AttributeError:
        decision_values = classifier.predict_proba(X_grid)[:, 1]
        levels = [.5]
        fill_levels = [0, .5, 1]

    if ax is None:
        ax = plt.gca()
    if fill:
        ax.contourf(x1,
                    x2,
                    decision_values.reshape(x1.shape),
```

```
                    levels=fill_levels,
                    colors=['cyan', 'pink', 'yellow'])
    if line:
        ax.contour(x1,
                   x2,
                   decision_values.reshape(x1.shape),
                   levels=levels,
                   colors='black')

    ax.set_xlim(x_min, x_max)
    ax.set_ylim(y_min, y_max)
    ax.set_xticks(())
    ax.set_yticks(())


# ## Генерация выборки

# In[4]:


X, y = make_classification(n_features=2,
                           n_samples=1000,
                           n_redundant=0,
                           n_informative=1,
                           n_clusters_per_class=1,
                           random_state=15,
                           class_sep=0.6)


# In[5]:


print('Координаты точек: ')
print(X[:15])
print('Метки класса: ')
print(y[:15])


#

# In[6]:


plt.scatter(X[:, 0], X[:, 1], c=y)
plt.show()


# ## Разбитие выборки на обучающее и тестовое множество

# In[7]:


X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    test_size=0.35,
                                                    random_state=42)


# In[8]:


plt.title('Обучающая выборка')
plt.scatter(X_train[:, 0], X_train[:, 1], c=y_train)
```

```
plt.show()


# In[9]:


plt.title('Тестовая выборка')
plt.scatter(X_test[:, 0], X_test[:, 1], c=y_test)
plt.show()


# ## Кластеризация

# In[10]:


from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.metrics import roc_auc_score

def show_info(classifier, classifier_name, real_values, prediction_values):
    print(f'Метод классификации: {classifier_name}\n')

    # Выводим предсказанное и реальное значение
    print('Предсказанные и реальные значения:')
    print(prediction_values)
    print(real_values)

    # Выводим матрицу неточностей
    print('\nМатрица неточностей')
    print(confusion_matrix(real_values, prediction_values))

    # Выводим точность классификации
    print(f'\nТочность классификации: {accuracy_score(prediction_values,
real_values)}')

    # Выводим полноту
    print('\nПолнота: ')
    print(classification_report(real_values, prediction_values))

    # AUC ROC
    print(f'\nПлощадь под кривой: {roc_auc_score(real_values,
prediction_values)}')

    plt.xlabel('Первый класс')
    plt.ylabel('Второй класс')
    plt.title(classifier_name.upper())
    plot_2d_separator(classifier, X, fill=True)
    plt.scatter(X[:, 0], X[:, 1], c=y, s=70)


# ### Метод k-ближайших соседей (1)

# In[11]:


from sklearn.neighbors import KNeighborsClassifier


# In[12]:
```

```
knn = KNeighborsClassifier(n_neighbors=1, metric='euclidean')

# Обучаем модель данных
knn.fit(X_train, y_train)

# Оцениваем качество модели
prediction = knn.predict(X_test)

# Выводим сводную информацию
show_info(knn, 'ближайшие соседи (1)', y_test, prediction)


# In[13]:


knn = KNeighborsClassifier(n_neighbors=3, metric='euclidean')

# Обучаем модель данных
knn.fit(X_train, y_train)

# Оцениваем качество модели
prediction = knn.predict(X_test)

# Выводим сводную информацию
show_info(knn, 'ближайшие соседи (3)', y_test, prediction)


# In[14]:


knn = KNeighborsClassifier(n_neighbors=5, metric='euclidean')

# Обучаем модель данных
knn.fit(X_train, y_train)

# Оцениваем качество модели
prediction = knn.predict(X_test)

# Выводим сводную информацию
show_info(knn, 'ближайшие соседи (5)', y_test, prediction)


# In[15]:


knn = KNeighborsClassifier(n_neighbors=9, metric='euclidean')

# Обучаем модель данных
knn.fit(X_train, y_train)

# Оцениваем качество модели
prediction = knn.predict(X_test)

# Выводим сводную информацию
show_info(knn, 'ближайшие соседи (9)', y_test, prediction)


# ## Наивный байесовский классификатор

# In[16]:


from sklearn.naive_bayes import GaussianNB
```

```
nb = GaussianNB()

# Обучаем модель данных
nb.fit(X_train, y_train)

# Оцениваем качество модели
prediction = nb.predict(X_test)

# Выводим сводную информацию
show_info(nb, 'Наивный байесовский классификатор', y_test, prediction)


# ## Случайный лес

# In[17]:


from sklearn.ensemble import RandomForestClassifier

rfc = RandomForestClassifier(n_estimators=5)

# Обучаем модель данных
rfc.fit(X_train, y_train)

# Оцениваем качество модели
prediction = rfc.predict(X_test)

# Выводим сводную информацию
show_info(rfc, 'случайный лес (5)', y_test, prediction)


# In[18]:


rfc = RandomForestClassifier(n_estimators=10)

# Обучаем модель данных
rfc.fit(X_train, y_train)

# Оцениваем качество модели
prediction = rfc.predict(X_test)

# Выводим сводную информацию
show_info(rfc, 'случайный лес (10)', y_test, prediction)


# In[19]:


rfc = RandomForestClassifier(n_estimators=15)

# Обучаем модель данных
rfc.fit(X_train, y_train)

# Оцениваем качество модели
prediction = rfc.predict(X_test)

# Выводим сводную информацию
show_info(rfc, 'случайный лес (15)', y_test, prediction)


# In[20]:
```

```
rfc = RandomForestClassifier(n_estimators=20)

# Обучаем модель данных
rfc.fit(X_train, y_train)

# Оцениваем качество модели
prediction = rfc.predict(X_test)

# Выводим сводную информацию
show_info(rfc, 'случайный лес (20)', y_test, prediction)


# In[21]:


rfc = RandomForestClassifier(n_estimators=50)

# Обучаем модель данных
rfc.fit(X_train, y_train)

# Оцениваем качество модели
prediction = rfc.predict(X_test)

# Выводим сводную информацию
show_info(rfc, 'случайный лес (50)', y_test, prediction)
```