Липецкий государственный технический университет

Факультет автоматизации и информатики

Кафедра автоматизированных систем управления

ЛАБОРАТОРНАЯ РАБОТА №2

по дисциплине

«Прикладные интеллектуальные системы и экспертные системы»

Студент Мамедов Р. В.

Группа М-ИАП-23-1

Руководитель Кургасов В. В.

доцент, канд. пед. наук

Липецк 2023 г.

Цель работы

Получить практические навыки решения задачи бинарной классификации данных в среде Jupiter Notebook. Научиться загружать данные, обучать классификаторы и проводить классификацию. Научиться оценивать точность полученных моделей.

Задание кафедры

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

2) Импортировать необходимые для работы библиотеки и модули

3) Загрузить данные в соответствие 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)

• Значения 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:

a) Метод к-ближайших соседей (n_neighbors = {1, 3, 5, 9})

b) Наивный байесовский метод

c) 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 качество классификации, если на тестовую часть выделить 10% выборки, 35% выборки. Для этого повторить п.п. 6 – 10.

Ход работы

На рисунке 1 представлены импортируемые необходимые библиотеки и модули для начальной работы программы. Остальные будут добавляться по мере выполнения работы.

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import make_classification
import numpy as np
```

Загружаем данные в соответствие с вариантом (вариант 12) и выводим первые 15 элементов выборки (рисунки 2-3).

```
ds = make_classification(random_state=23, n_informative=1, n_redundant=0,
n_features=2, n_clusters_per_class=1,
                         class_sep=0.35)
df = pd.DataFrame(ds[0], columns=['first', 'second'])
df['res'] = ds[1]
df.head(15)
```

	first	second	class
0	-2.524672	5.509991	0
1	-13.433619	-1.698099	1
2	-3.429146	0.204845	1
3	1.869207	3.301294	0
4	-11.681285	-6.357410	1
5	-11.214032	4.703204	1
6	4.264361	9.229041	0
7	-4.744878	-0.153989	1
8	2.050991	-1.903319	0
9	4.388322	1.433953	0
10	1.431567	1.463239	0
11	2.677893	0.997666	0
12	-11.369388	-4.656133	1
13	5.634561	6.802091	0
14	4.425509	-2.606092	0

Рисунок 1 – Первые 15 элементов выборки

Отобразим на графике сгенерированную выборку (рисунок 2).

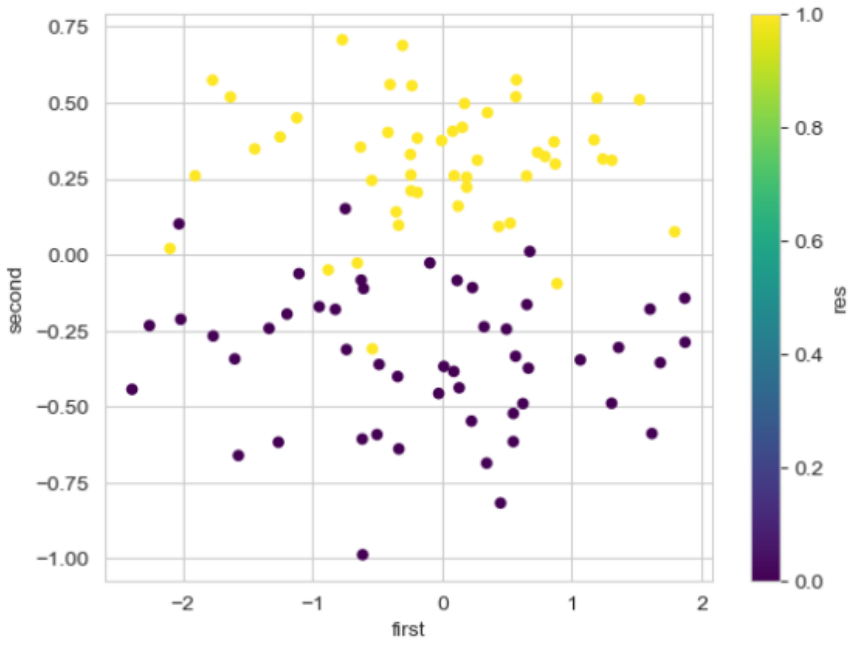

Рисунок 2 – Сгенерированная выборка

Разобьем данные на обучающую и тестовую выборки.

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(ds[0], ds[1])
```

Отобразим на графике обучающую выборку (рисунок 3).

```
train_df = pd.DataFrame(x_train, columns=['first', 'second'])
train_df['res'] = y_train
train_df.plot.scatter(x='first', y='second', c='res', colormap='viridis')
```

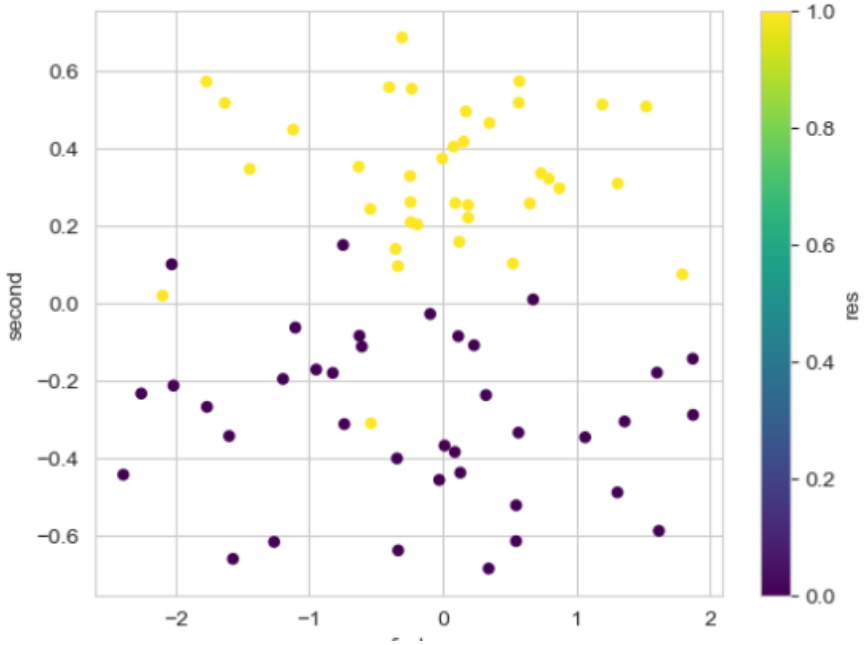

Рисунок 3– Обучающая выборка (код и график)

Отобразим на графике тестовую выборку (рисунок 4).

```
test_df = pd.DataFrame(x_test, columns=['first', 'second'])
test_df['res'] = y_test
test_df.plot.scatter(x='first', y='second', c='res', colormap='viridis')
```

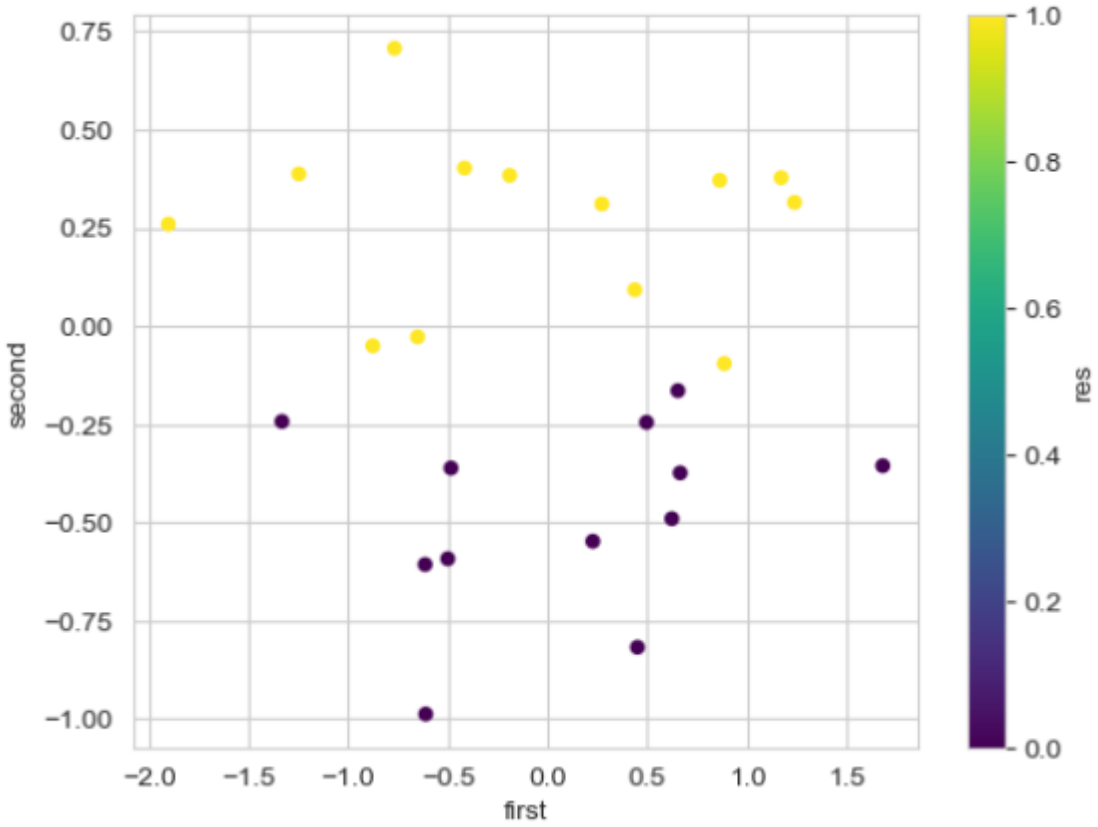

Рисунок 4 – тестовая выборка

Функция для вывода результатов классификации:

```
def show_statistic(clf, x_test, y_test):
    clf.fit(x_train, y_train)
    print(f"y_true: {y_test}")
    y_pred = clf.predict(x_test)
    print(f"y_pred: {y_pred}")
    cm = confusion_matrix(y_test, y_pred, labels=clf.classes_)
    disp = ConfusionMatrixDisplay(confusion_matrix=cm,
display_labels=clf.classes_)
    disp.plot()
    print(classification_report(y_test, y_pred, target_names=['first',
'second']))
    print("area under curve: {:.2f}".format(roc_auc_score(y_test, y_pred)))
    DecisionBoundaryDisplay.from_estimator(clf, x_test)
```

Напишем функцию, реализующую метод k-ближних соседей с принимаемыми параметрами.

```
def test_KNeighthboursClassifier_hyper(hyperparams):
    for param in hyperparams:
        print(f"param = {param}")
        clf = KNeighborsClassifier(n_neighbors=param)
```

```
        show_statistic(clf, x_test, y_test)

test_KNeighthboursClassifier_hyper([1, 3, 5, 9])
```

Выполним данную функцию с параметрами n_neighbors = 1, 3, 5 и 9 и выведем результаты.

```
param = 1
Prediction and test:
prediction: [0 0 1 1 0 0 1 1 0 1 1 1 1 0 0 1 0 0 1 1 0 0 0 1 0]
y_test: [0 0 1 1 0 0 0 0 0 1 1 1 1 0 0 1 0 0 1 1 0 0 0 1 0]
              precision    recall  f1-score   support

       first       1.00      0.87      0.93        15
      second       0.83      1.00      0.91        10

    accuracy                           0.92        25
   macro avg       0.92      0.93      0.92        25
weighted avg       0.93      0.92      0.92        25

AUC ROC: 0.93
```

Рисунок 5 – Статистика по методу k-ближних соседей (n_neighbors = 1)

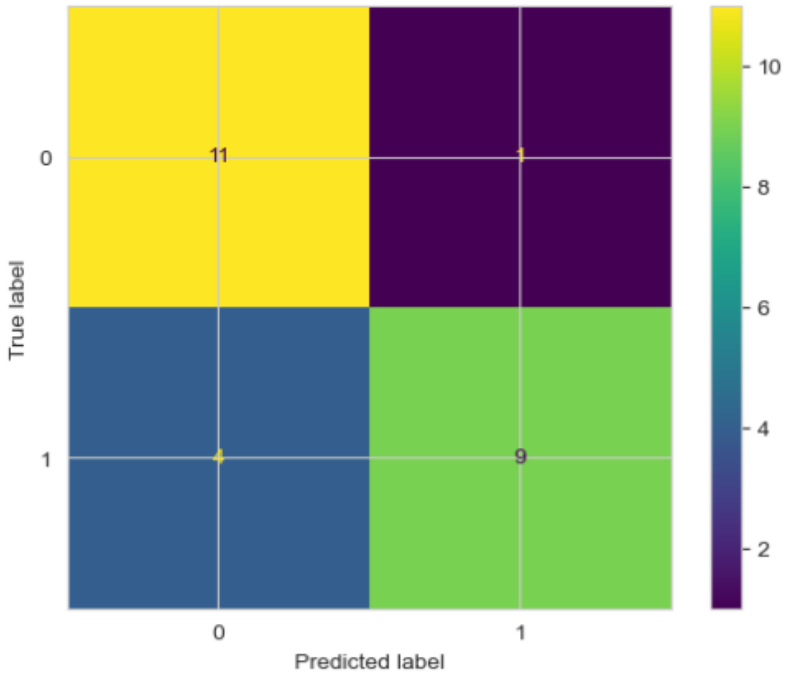

Рисунок 6 – Матрица ошибок по методу k-ближних соседей (n_neighbors = 1)

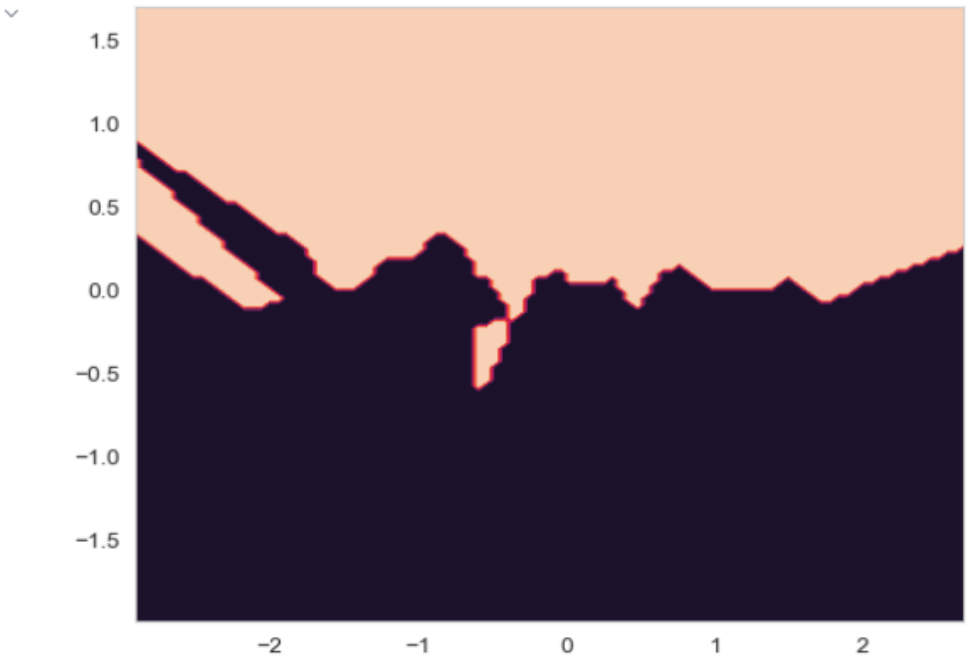

Рисунок 7 – Области принятия решений (метод k-ближних соседей (n_neighbors = 1))

Проведём классификацию наивным байесовским методом с параметрами по умолчанию. Код представлен на рисунке 22.

```
from sklearn.naive_bayes import GaussianNB

clf = GaussianNB()
clf.fit(x_train, y_train)
show_statistic(clf, x_test, y_test)
```

```
y_true: [0 0 1 0 1 1 0 1 1 1 0 0 0 1 1 1 1 0 0 1 0 1 0 0 1]
y_pred: [0 0 0 0 1 1 0 1 0 1 0 0 0 1 1 1 1 0 0 1 0 0 0 0 1]
              precision    recall  f1-score   support

       first       0.80      1.00      0.89        12
      second       1.00      0.77      0.87        13

    accuracy                           0.88        25
   macro avg       0.90      0.88      0.88        25
weighted avg       0.90      0.88      0.88        25

area under curve: 0.88
```

Рисунок 8 – Статистика по наивному байесовскому методу

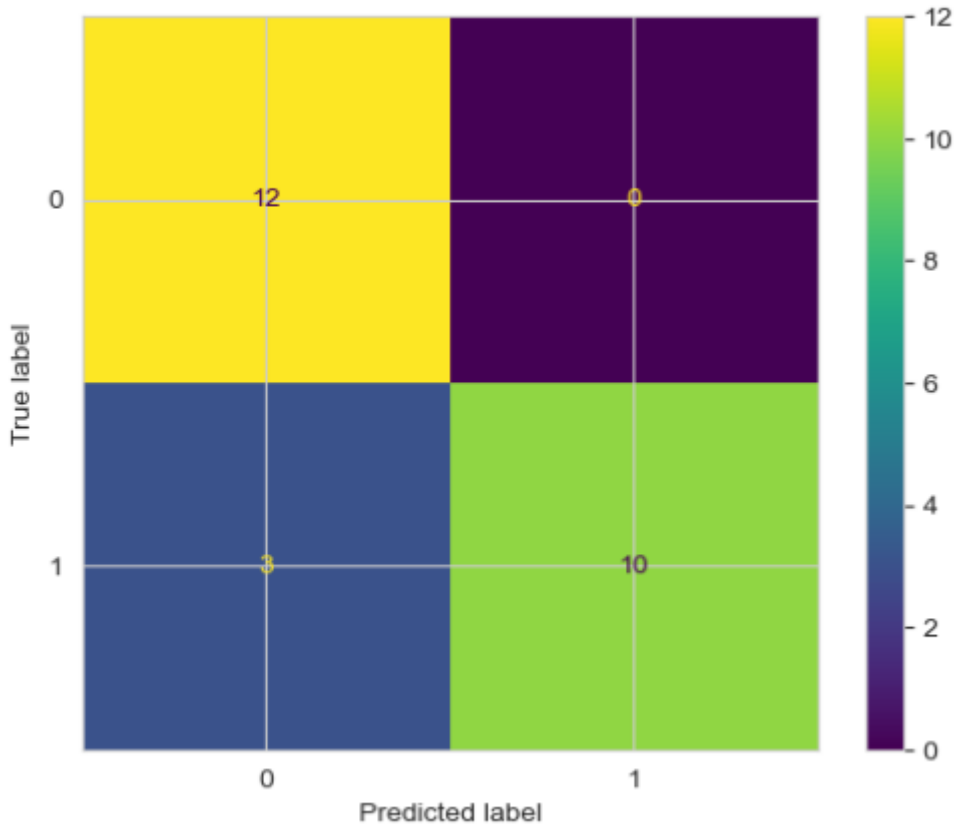

Рисунок 9 – Матрица ошибок по наивному байесовскому методу

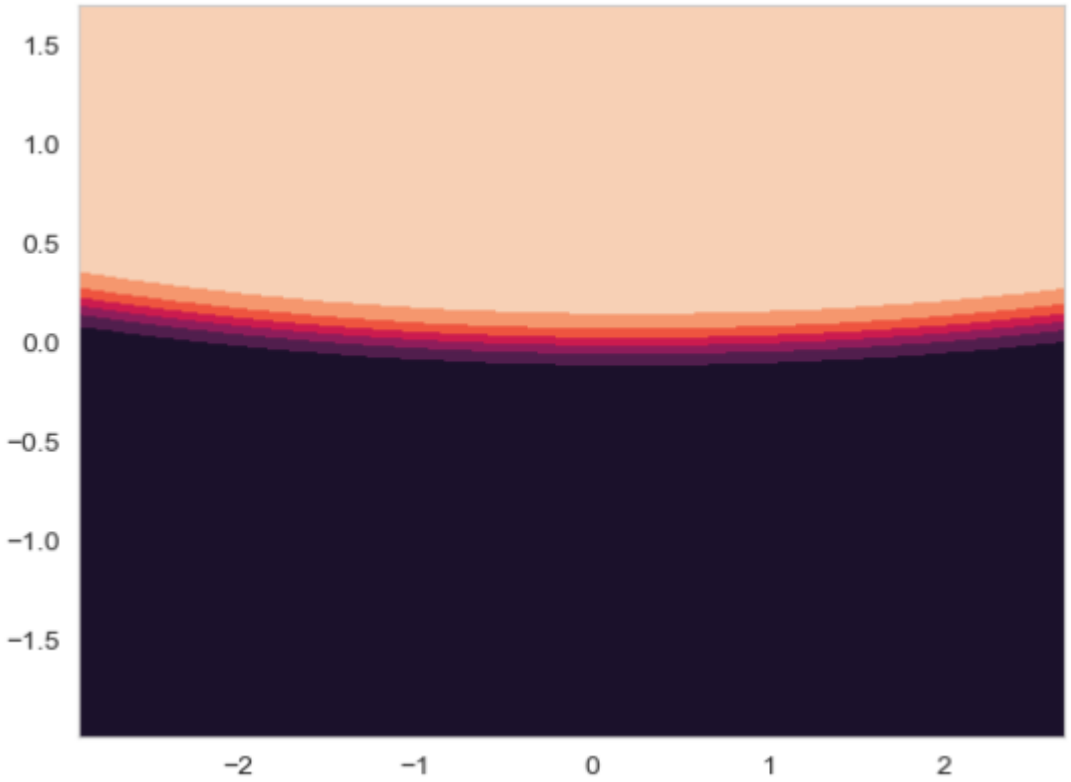

Рисунок 10 – Области принятия решения по наивному байесовскому методу

Проведём классификацию методом случайного леса поочередно используя параметр n_estimators = 5, 10, 15, 20, 50. Данная функция на языке python имеет следующий вид:

```
from sklearn.ensemble import RandomForestClassifier

def test_RandomForestClassifier_hyper(hyperparams):
    for param in hyperparams:
        print(f"param = {param}")
        clf = RandomForestClassifier(n_estimators=param)
        clf.fit(x_train, y_train)
        show_statistic(clf, x_test, y_test)
```

Результаты работы представлены на рисунках 11-13.

```
param = 5
y_true: [0 0 1 0 1 1 0 1 1 1 0 0 0 1 1 1 1 0 0 1 0 1 0 0 1]
y_pred: [0 0 0 0 1 1 0 1 0 1 0 0 0 1 1 1 1 0 0 1 0 0 0 0 1]
              precision    recall  f1-score   support

       first       0.80      1.00      0.89        12
      second       1.00      0.77      0.87        13

    accuracy                           0.88        25
   macro avg       0.90      0.88      0.88        25
weighted avg       0.90      0.88      0.88        25

area under curve: 0.88
```

Рисунок 11 – Статистика по методу случайного леса (n_estimators = 5)

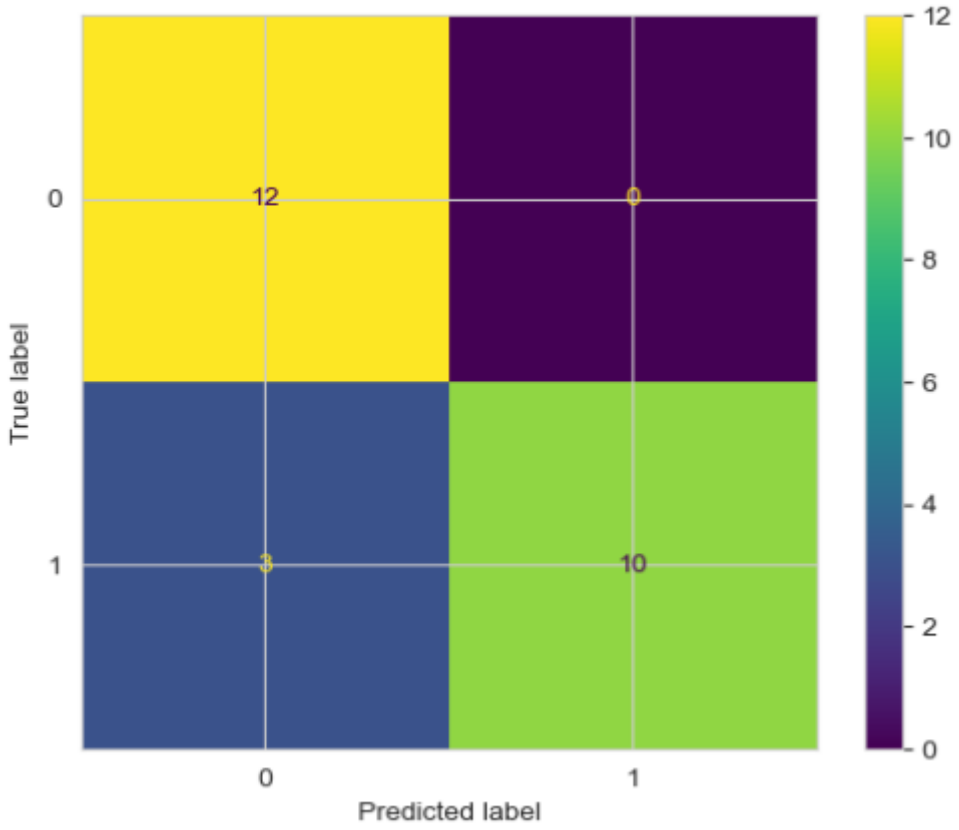

Рисунок 12 – Матрица ошибок по методу случайного леса (n_estimators = 5)

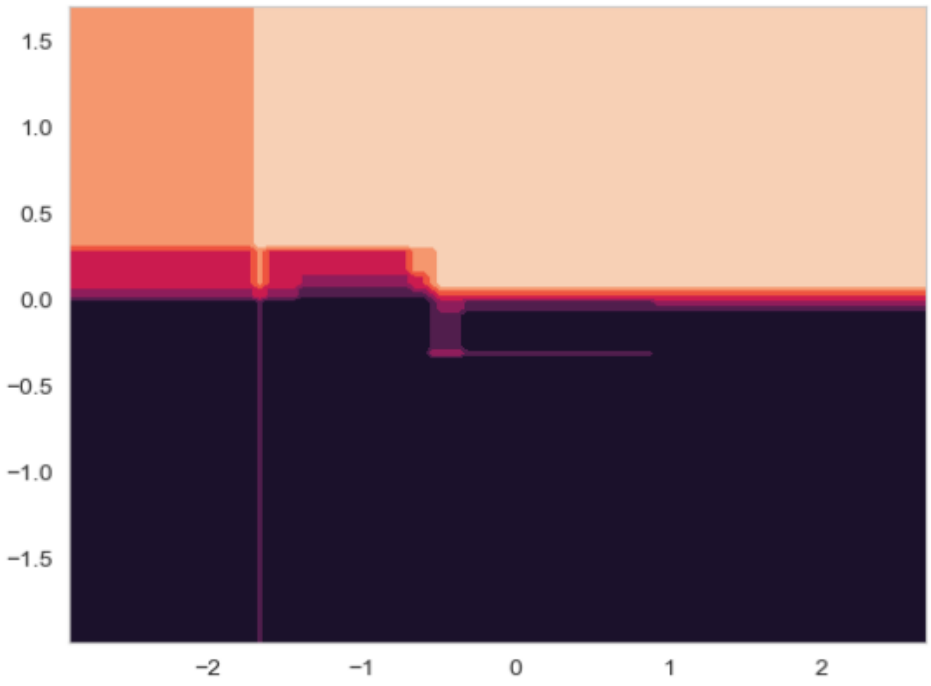

Рисунок 13 – Области принятия решений по методу случайного леса (n_estimators = 5)

По итогам классификации при размере тестововой выборки 25% лучшие результаты показал метод случайного леса (n_estimators = 5,15,20,50) и метод k-ближних соседей (n_neighbors = 3,5,9). Увеличение параметров в этих методах не ведёт к улучшению результатов, а в случае с методом случайного леса вызывает ухудшение классификации. Увеличение тестовой выборки особо не ведёт к улучшению показателей (кроме наивного байесовского). Заполним сводную таблицу (таблица 1) для каждого метода и размера тестовой выборки 10%, 25% и 35%.

Таблица 1 – Сводная таблица

Метод	Размер тестовой выборки	Результат
k-ближних	10 %	Confusion matrix: [6 0] [0 4] Accuracy score: 0.91 AUC ROC:0,92
Наивный байесовский		Confusion matrix: [6 0] [0 4] Accuracy score: 0.94 AUC ROC: 0.95
Рандомный лес		Confusion matrix: [6 0] [0 4] Accuracy score: 0.94 AUC ROC: 0.95
k-ближних	25%	Confusion matrix: [11 1] [14 9] Accuracy score: 0,91 AUC ROC:0,92
Наивный байесовский		Confusion matrix: [12 0] [3 10] Accuracy score: 0,94 AUC ROC: 0,95
Рандомный лес		Confusion matrix: [12 0] [3 10] Accuracy score: 0,94 AUC ROC: 0,95

Окончание таблицы 1

Метод	Размер тестовой выборки	Результат
k-ближних	35%	Confusion matrix: [16 2] [0 17] Accuracy score: 0,94 AUC ROC:0,94
Наивный байесовский		Confusion matrix: [16 2] [0 17] Accuracy score: 0,94 AUC ROC: 0,94
Рандомный лес		Confusion matrix: [16 2] [0 17] Accuracy score: 0,94 AUC ROC: 0,94

Вывод

Мы получили практические навыки решения задачи бинарной классификации данных.

Приложение А (рекомендованное)

Исходный код

```
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.datasets import make_blobs


ds = make_blobs(random_state=28, centers=2, cluster_std=4.5,
shuffle=1)
df = pd.DataFrame(ds[0], columns=['first', 'second'])
df['class'] = ds[1]
df.head(15)


df.plot.scatter(x='first', y='second', c='class',
colormap='viridis')


from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(ds[0],
ds[1])


train_df = pd.DataFrame(x_train, columns=['first', 'second'])
train_df['class'] = y_train
train_df.plot.scatter(x='first', y='second', c='class',
colormap='viridis')

test_df = pd.DataFrame(x_test, columns=['first', 'second'])
test_df['class'] = y_test
test_df.plot.scatter(x='first', y='second', c='class',
colormap='viridis')

from sklearn.metrics import confusion_matrix,
ConfusionMatrixDisplay, classification_report, roc_auc_score
from sklearn.inspection import DecisionBoundaryDisplay

def show_statistic(knn, x_test, y_test):
    knn.fit(x_train, y_train)
    prediction = knn.predict(x_test)

    print('Prediction and test: ')
```

```
    print(f"prediction: {prediction}")
    print(f"y_test: {y_test}")

    cm = confusion_matrix(y_test, prediction,
labels=knn.classes_)
    disp = ConfusionMatrixDisplay(confusion_matrix=cm,
display_labels=knn.classes_)
    disp.plot()
    print(classification_report(y_test, prediction,
target_names=['first', 'second']))
    print("AUC ROC: {:.2f}".format(roc_auc_score(y_test,
prediction)))
    disp = DecisionBoundaryDisplay.from_estimator(knn, ds[0],
response_method="predict")
    disp.ax_.scatter(ds[0][:,0], ds[0][:,1], c=ds[1],
edgecolor="k")
    plt.show()


from sklearn.neighbors import KNeighborsClassifier

def test_KNeighthboursClassifier(neighbors_params):
    for param in neighbors_params:
        print(f"param = {param}")
        clf = KNeighborsClassifier(n_neighbors=param)
        show_statistic(clf, x_test, y_test)

test_KNeighthboursClassifier([1, 3, 5, 9])


from sklearn.naive_bayes import GaussianNB

knn = GaussianNB()
knn.fit(x_train, y_train)
show_statistic(knn, x_test, y_test)


from sklearn.ensemble import RandomForestClassifier

def test_RandomForestClassifier(estimators_params):
    for param in estimators_params:
        print(f"param = {param}")
        clf = RandomForestClassifier(n_estimators=param)
```

```
        clf.fit(x_train, y_train)
        show_statistic(clf, x_test, y_test)

test_RandomForestClassifier([5, 10, 15, 20, 50])


x_train, x_test, y_train, y_test = train_test_split(ds[0],
ds[1], test_size=0.1)

train_df = pd.DataFrame(x_train, columns=['first', 'second'])
train_df['class'] = y_train
train_df.plot.scatter(x='first', y='second', c='class',
colormap='viridis')


test_df = pd.DataFrame(x_test, columns=['first', 'second'])
test_df['class'] = y_test
test_df.plot.scatter(x='first', y='second', c='class',
colormap='viridis')


test_KNeighthboursClassifier([1, 3, 5, 9])

knn = GaussianNB()
knn.fit(x_train, y_train)
show_statistic(knn, x_test, y_test)

test_RandomForestClassifier([5, 10, 15, 20, 50])


x_train, x_test, y_train, y_test = train_test_split(ds[0],
ds[1], test_size=0.35)

train_df = pd.DataFrame(x_train, columns=['first', 'second'])
train_df['class'] = y_train
train_df.plot.scatter(x='first', y='second', c='class',
colormap='viridis')


test_df = pd.DataFrame(x_test, columns=['first', 'second'])
test_df['class'] = y_test
test_df.plot.scatter(x='first', y='second', c='class',
colormap='viridis')
```

```
test_KNeighthboursClassifier([1, 3, 5, 9])

knn = GaussianNB()
knn.fit(x_train, y_train)
show_statistic(knn, x_test, y_test)

test_RandomForestClassifier([5, 10, 15, 20, 50])
```