Лосева Елизавета Юрьевна, группа 8-2

Лабораторная работа № 7

Вариант № 4

Распознавание образов с использованием машины опорных векторов

Цель работы:

Исследовать алгоритмы распознавания образов на основе аппарата машины опорных векторов (Support Vector Machine).

Задание:

Получить у преподавателя вариант задания и написать код, реализующий соответствующий алгоритм обработки информации. Для ответа на поставленные в задании вопросы провести численный эксперимент или статистическое имитационное моделирование и представить соответствующие графики. Провести анализ полученных результатов и представить его в виде выводов по проделанной работе.

Задание по варианту:

Воспользовавшись классификатором SVM, определите вероятности ошибок классификации линейно НЕразделимых выборок двух классов для следующих типов ядер: квадратичная функция, полиномиальная функция. Определите оптимальную функцию ядра.

Код программы (внесённые изменения в шаблон кода выделены)

```
import matplotlib
import numpy as np
matplotlib.use('TkAgg')
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
from sklearn import datasets, metrics, model_selection, svm
from collections import namedtuple
import seaborn as sns
import sys
import io

sys.stdout = io.TextIOWrapper(sys.stdout.buffer, encoding='utf-8')

# Настройка стиля графиков
sns.set_style("darkgrid")
plt.rcParams['figure.facecolor'] = 'white'
plt.rcParams['axes.facecolor'] = '#f8f9fa'

# Конфигурация ядра SVM
KernelSettings = namedtuple('KernelSettings', ['name', 'hyperparameters'])


class SVMAnalyzer:
    # Класс для анализа и сравнения различных ядер SVM
```

```
    def __init__(self, data_X, data_y):
        self.X = data_X
        self.y = data_y
        self.models = []

    def create_dataset(self, samples=1000, noise_level=0.25, seed=42):
        # Создание набора данных make_moons
        X, y = datasets.make_moons(
            n_samples=samples,
            noise=noise_level,
            random_state=seed
        )
        self.X = X
        self.y = y
        return X, y

    def single_experiment(self, kernel_setup, test_portion=0.3, seed=None):
        # Проведение одного эксперимента с заданным ядром
        X_tr, X_te, y_tr, y_te = model_selection.train_test_split(
            self.X, self.y,
            test_size=test_portion,
            stratify=self.y,
            random_state=seed
        )

        classifier = svm.SVC(**kernel_setup.hyperparameters)
        classifier.fit(X_tr, y_tr)
        predictions = classifier.predict(X_te)

        error_rate = 1.0 - metrics.accuracy_score(y_te, predictions)
        return error_rate

    def monte_carlo_analysis(self, kernel_configs, test_portion=0.3,
iterations=50):
        # Монте-Карло анализ для оценки стабильности моделей
        analysis_results = []
        np.random.seed(1234)
        random_seeds = np.random.randint(0, 10000, size=iterations)

        for kernel_cfg in kernel_configs:
            error_rates = []
            for seed_val in random_seeds:
                err = self.single_experiment(
                    kernel_cfg,
                    test_portion=test_portion,
                    seed=int(seed_val)
                )
                error_rates.append(err)

            error_array = np.array(error_rates)
            analysis_results.append({
```

```
                'kernel_name': kernel_cfg.name,
                'avg_error': np.mean(error_array),
                'error_std': np.std(error_array, ddof=1),
                'lowest_error': np.min(error_array),
                'highest_error': np.max(error_array),
                'avg_accuracy': 1.0 - np.mean(error_array),
                'all_errors': error_array
            })

        return analysis_results

    def train_and_evaluate(self, kernel_configs, test_portion=0.3, seed=2024):
        # Обучение моделей и сбор метрик
        X_tr, X_te, y_tr, y_te = model_selection.train_test_split(
            self.X, self.y,
            test_size=test_portion,
            stratify=self.y,
            random_state=seed
        )

        model_reports = []
        for kernel_cfg in kernel_configs:
            classifier = svm.SVC(**kernel_cfg.hyperparameters)
            classifier.fit(X_tr, y_tr)
            predictions = classifier.predict(X_te)

            confusion_mtx = metrics.confusion_matrix(y_te, predictions)
            class_report = metrics.classification_report(
                y_te, predictions,
                target_names=["Класс 0", "Класс 1"],
                output_dict=True,
                zero_division=0
            )

            model_reports.append({
                'kernel_name': kernel_cfg.name,
                'confusion_matrix': confusion_mtx,
                'classification_report': class_report,
                'accuracy': metrics.accuracy_score(y_te, predictions),
                'f1_macro': class_report["macro avg"]["f1-score"],
                'classifier': classifier
            })

        return model_reports, X_tr, X_te, y_tr, y_te


def visualize_data_distribution(X, y):
    # Визуализация исходного распределения данных
    fig, ax = plt.subplots(figsize=(7, 6))

    scatter = ax.scatter(
```

```
        X[:, 0], X[:, 1],
        c=y,
        cmap='coolwarm',  # ИЗМЕНЕНО
        s=50,
        alpha=0.7,
        edgecolors='black',
        linewidths=0.5
    )

    ax.set_title("Распределение данных (make_moons)", fontsize=14,
fontweight='bold')
    ax.set_xlabel("Признак X1", fontsize=12)
    ax.set_ylabel("Признак X2", fontsize=12)
    ax.grid(True, alpha=0.3, linestyle=':')

    plt.tight_layout()
    plt.show()


def visualize_error_comparison(analysis_results):
    # Визуализация сравнения ошибок классификации
    kernel_names = [r['kernel_name'] for r in analysis_results]
    mean_errors = [r['avg_error'] for r in analysis_results]
    error_stds = [r['error_std'] for r in analysis_results]

    fig, ax = plt.subplots(figsize=(10, 6))

    x_pos = np.arange(len(kernel_names))
    colors = ['#FF1493', '#00CED1', '#FFD700']

    bars = ax.barh(
        x_pos, mean_errors,
        xerr=error_stds,
        capsize=8,
        color=colors[:len(kernel_names)],
        alpha=0.8,
        edgecolor='black',
        linewidth=1.5
    )

    # Добавление значений на столбцы
    for i, (bar, err, std) in enumerate(zip(bars, mean_errors, error_stds)):
        ax.text(
            err + std + 0.005,
            bar.get_y() + bar.get_height() / 2,
            f'{err:.3f} ± {std:.3f}',
            va='center',
            fontsize=10,
            fontweight='bold'
        )
```

```
    ax.set_yticks(x_pos)
    ax.set_yticklabels(kernel_names, fontsize=11)
    ax.set_xlabel("Средняя вероятность ошибки", fontsize=12, fontweight='bold')
    ax.set_title("Сравнение качества классификации различных ядер SVM",
                 fontsize=13, fontweight='bold', pad=15)
    ax.set_xlim(0, max(mean_errors) + max(error_stds) + 0.05)
    ax.grid(axis='x', alpha=0.3, linestyle='--')
    ax.invert_yaxis()

    plt.tight_layout()
    plt.show()


def visualize_confusion_matrices(model_reports):
    # Визуализация матриц ошибок для всех моделей
    num_models = len(model_reports)
    cols = min(3, num_models)
    rows = (num_models + cols - 1) // cols

    fig, axes = plt.subplots(rows, cols, figsize=(6 * cols, 5 * rows))
    if num_models == 1:
        axes = np.array([axes])
    axes = axes.flatten() if num_models > 1 else axes

    for idx, report in enumerate(model_reports):
        ax = axes[idx] if num_models > 1 else axes
        cm = report['confusion_matrix']

        # Используем тепловую карту
        sns.heatmap(
            cm,
            annot=True,
            fmt='d',
            cmap='Blues',
            cbar=True,
            ax=ax,
            square=True,
            linewidths=2,
            linecolor='black'
        )

        ax.set_title(f"{report['kernel_name']}\nТочность:
{report['accuracy']:.3f}",
                     fontsize=11, fontweight='bold')
        ax.set_xlabel("Предсказанный класс", fontsize=10)
        ax.set_ylabel("Истинный класс", fontsize=10)
        ax.set_xticklabels(["Класс 0", "Класс 1"])
        ax.set_yticklabels(["Класс 0", "Класс 1"])

    # Скрываем лишние подграфики
    for idx in range(num_models, len(axes)):
```

```
        axes[idx].axis('off')

    fig.suptitle("Матрицы ошибок классификации", fontsize=14, fontweight='bold',
y=1.0)
    plt.tight_layout()
    plt.show()


def visualize_decision_surfaces(model_reports, X, y):
    # Визуализация поверхностей принятия решений
    num_models = len(model_reports)
    cols = min(3, num_models)
    rows = (num_models + cols - 1) // cols

    x_range = (X[:, 0].min() - 0.5, X[:, 0].max() + 0.5)
    y_range = (X[:, 1].min() - 0.5, X[:, 1].max() + 0.5)

    grid_x = np.linspace(x_range[0], x_range[1], 300)
    grid_y = np.linspace(y_range[0], y_range[1], 300)
    xx, yy = np.meshgrid(grid_x, grid_y)

    fig, axes = plt.subplots(rows, cols, figsize=(6 * cols, 5.5 * rows))
    if num_models == 1:
        axes = np.array([axes])
    axes = axes.flatten() if num_models > 1 else axes

    for idx, report in enumerate(model_reports):
        ax = axes[idx] if num_models > 1 else axes
        clf = report['classifier']

        grid_points = np.c_[xx.ravel(), yy.ravel()]
        Z = clf.predict(grid_points).reshape(xx.shape)

        # Используем другой стиль контуров
        ax.contourf(xx, yy, Z, levels=[-0.5, 0.5, 1.5], colors=['#FFCCCC',
'#CCFFCC'], alpha=0.4)  # ИЗМЕНЕНО
        ax.contour(xx, yy, Z, levels=[0.5], colors='red', linewidths=2,
linestyles='--')  # ИЗМЕНЕНО

        # Разные маркеры для разных классов
        class_0_mask = y == 0
        class_1_mask = y == 1

        ax.scatter(
            X[class_0_mask, 0], X[class_0_mask, 1],
            c='#8A2BE2', marker='o', s=40, alpha=0.8,  # ИЗМЕНЕНО
            edgecolors='black', linewidths=1, label='Класс 0'
        )
        ax.scatter(
            X[class_1_mask, 0], X[class_1_mask, 1],
            c='#FF6347', marker='s', s=40, alpha=0.8,  # ИЗМЕНЕНО
```

```
            edgecolors='black', linewidths=1, label='Класс 1'
        )

        ax.set_title(
            f"{report['kernel_name']}\nТочность: {report['accuracy']:.3f}",
            fontsize=11, fontweight='bold'
        )
        ax.set_xlabel("Признак X1", fontsize=10)
        ax.set_ylabel("Признак X2", fontsize=10)
        ax.grid(True, alpha=0.3)
        ax.legend(loc='upper right', fontsize=9)

    # Скрываем лишние подграфики
    for idx in range(num_models, len(axes)):
        axes[idx].axis('off')

    fig.suptitle("Поверхности принятия решений для различных ядер",
                 fontsize=14, fontweight='bold', y=1.0)
    plt.tight_layout()
    plt.show()


def main():
    # Основная функция выполнения эксперимента
    # Создание анализатора
    analyzer = SVMAnalyzer(None, None)
    X, y = analyzer.create_dataset()
    analyzer.X = X
    analyzer.y = y

    visualize_data_distribution(X, y)

    # Конфигурации ядер
    kernel_configurations = [
        KernelSettings(
            name="Полиномиальное ядро (степень 2)",
            hyperparameters={"kernel": "poly", "degree": 2, "coef0": 1.0, "C":
1.0, "gamma": "scale"}
        ),
        KernelSettings(
            name="Полиномиальное ядро (степень 3)",
            hyperparameters={"kernel": "poly", "degree": 3, "coef0": 1.0, "C":
1.0, "gamma": "scale"}
        ),
        KernelSettings(
            name="Полиномиальное ядро (степень 4)",
            hyperparameters={"kernel": "poly", "degree": 4, "coef0": 1.0, "C":
1.0, "gamma": "scale"}
        ),
    ]
```

```
    # Разделение на обучающую и тестовую выборки
    X_train, X_test, y_train, y_test = model_selection.train_test_split(
        X, y,
        test_size=0.3,
        stratify=y,
        random_state=2024
    )

    # Обучение и оценка моделей
    model_reports, _, _, _, _ =
analyzer.train_and_evaluate(kernel_configurations)

    # Монте-Карло анализ
    mc_results = analyzer.monte_carlo_analysis(kernel_configurations,
iterations=40)
    visualize_error_comparison(mc_results)

    # Вывод результатов
    print("\n" + "="*70)
    print("РЕЗУЛЬТАТЫ ЭКСПЕРИМЕНТА (средние значения по 40 прогонам Монте-
Карло)")
    print("="*70)
    for result in mc_results:
        print(f"\n{result['kernel_name']}:")
        print(f"  Средняя ошибка: {result['avg_error']:.4f} ±
{result['error_std']:.4f}")
        print(f"  Диапазон: [{result['lowest_error']:.4f},
{result['highest_error']:.4f}]")
        print(f"  Средняя точность: {result['avg_accuracy']:.4f}")

    # Определение лучшего ядра
    best_kernel = min(mc_results, key=lambda x: x['avg_error'])
    print(f"\n{'='*70}")
    print(f"ЛУЧШЕЕ ЯДРО: {best_kernel['kernel_name']}")
    print(f"{'='*70}")

    # Детальные метрики для лучшего ядра
    best_report = next(r for r in model_reports if r['kernel_name'] ==
best_kernel['kernel_name'])
    print(f"\nМетрики на фиксированном тестовом множестве:")
    print(f"  Accuracy: {best_report['accuracy']:.4f}")
    print(f"  F1-score (macro): {best_report['f1_macro']:.4f}")
    print(f"\nДетальный классификационный отчет:")
    report_dict = best_report['classification_report']
    for class_name in ["Класс 0", "Класс 1"]:
        metrics_dict = report_dict[class_name]
        print(f"  {class_name}:")
        print(f"    Precision: {metrics_dict['precision']:.3f}")
        print(f"    Recall: {metrics_dict['recall']:.3f}")
        print(f"    F1-score: {metrics_dict['f1-score']:.3f}")
```

```
    # Визуализация результатов
    visualize_confusion_matrices(model_reports)
    visualize_decision_surfaces(model_reports, X, y)


if __name__ == "__main__":
    main()
```

Результаты выполнения задания:

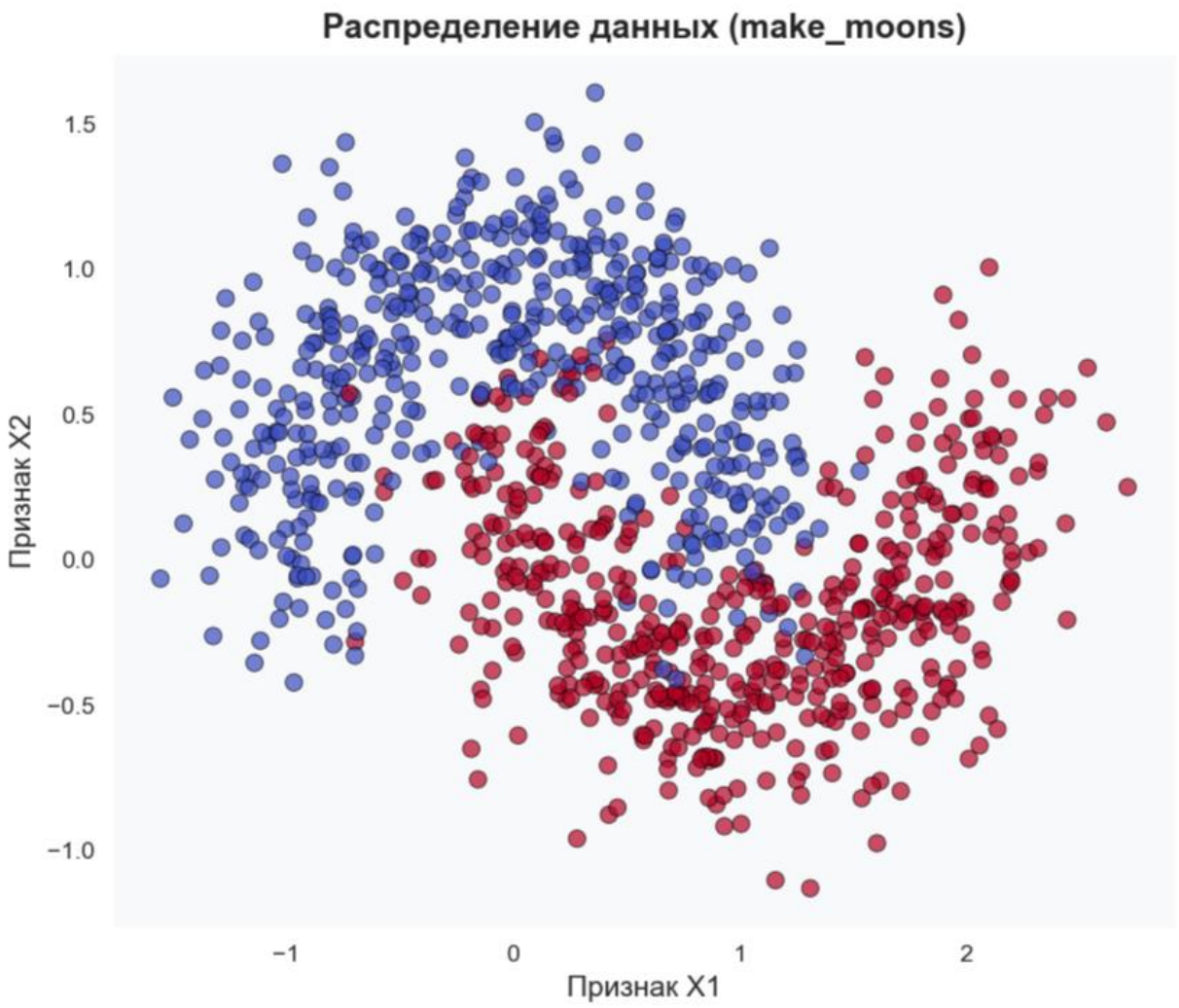

Рисунок 1.

Рисунок 2.

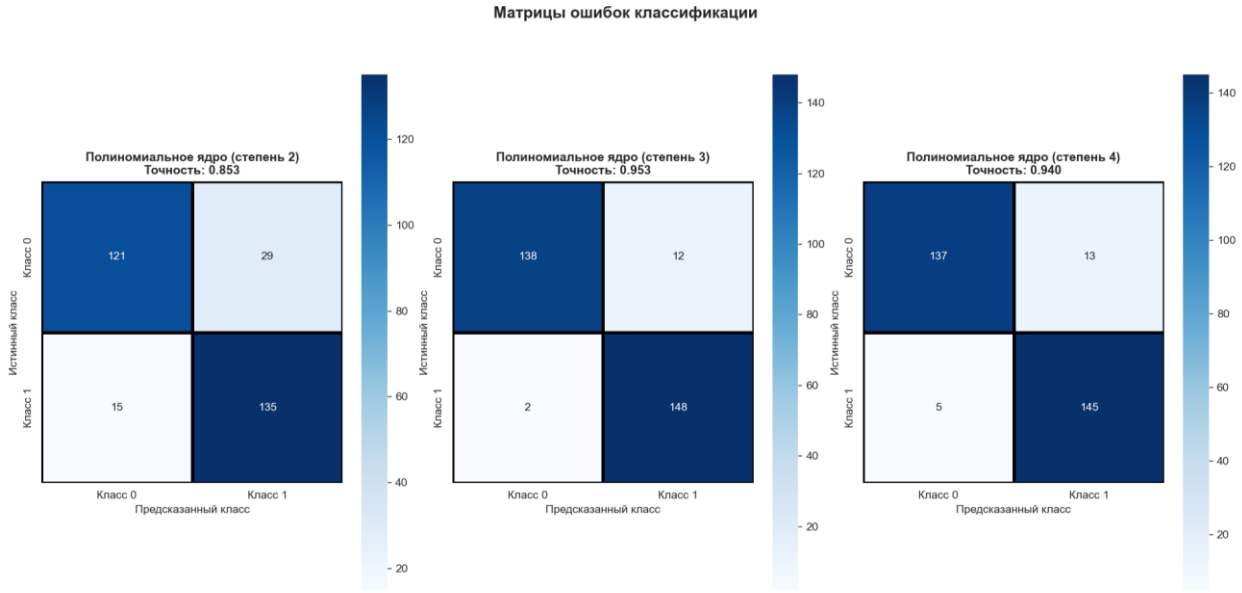

Рисунок 3.

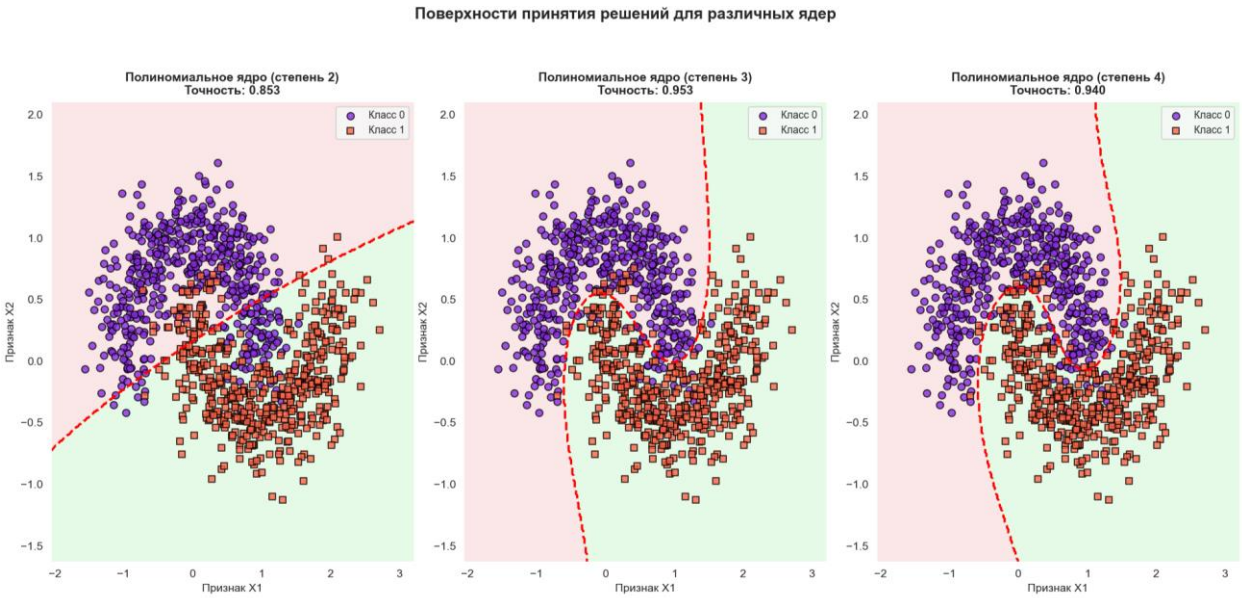

Рисунок 4.

```
================================================================================
РЕЗУЛЬТАТЫ ЭКСПЕРИМЕНТА (средние значения по 40 прогонам Монте-Карло)
================================================================================

Полиномиальное ядро (степень 2):
  Средняя ошибка: 0.1467 ± 0.0171
  Диапазон: [0.0900, 0.1767]
  Средняя точность: 0.8533

Полиномиальное ядро (степень 3):
  Средняя ошибка: 0.0540 ± 0.0109
  Диапазон: [0.0333, 0.0800]
  Средняя точность: 0.9460

Полиномиальное ядро (степень 4):
  Средняя ошибка: 0.0538 ± 0.0111
  Диапазон: [0.0333, 0.0800]
  Средняя точность: 0.9462
```

Рисунок 5.

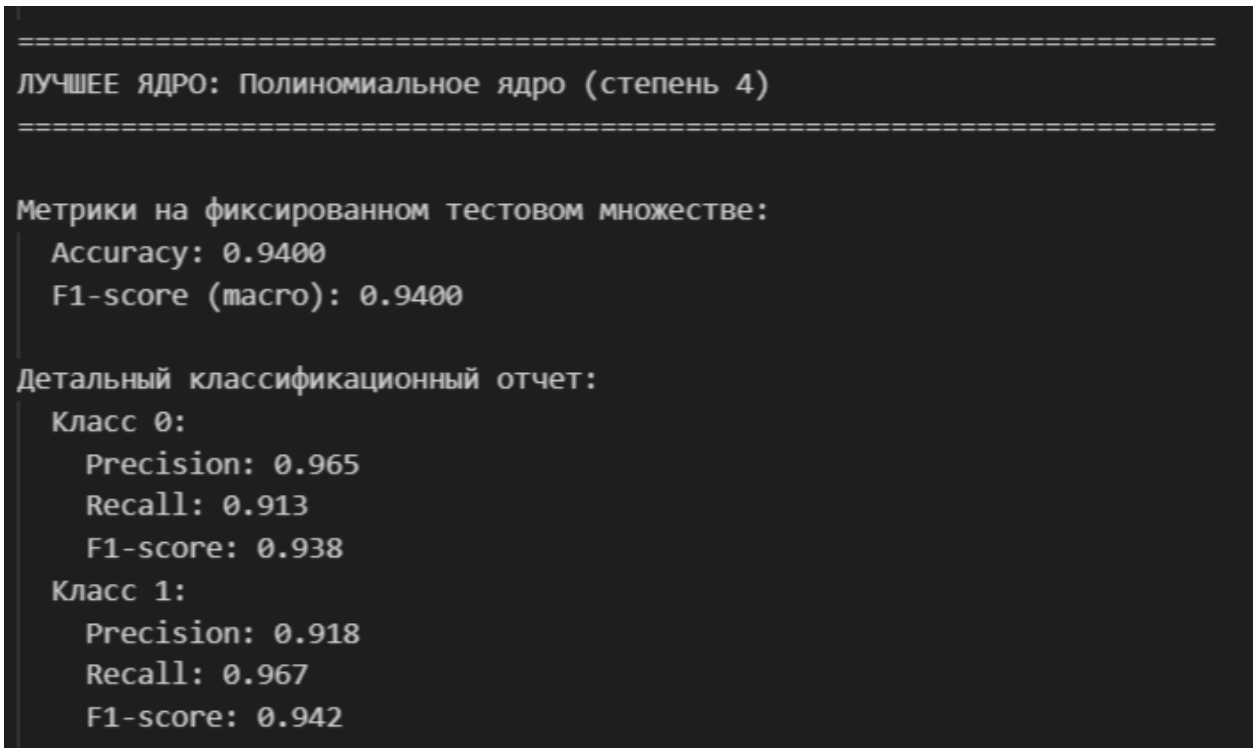

Рисунок 6.

Ответы на контрольные вопросы:

1. Параметр регуляризации C в методе SVM определяет баланс между точностью классификации и способностью модели к обобщению. При малом значении C модель является более мягкой и допускает значительное количество ошибок на обучающей выборке, чтобы добиться максимально широкой разделяющей полосы. Такой подход повышает устойчивость к шуму в данных и улучшает обобщающую способность, однако может привести к чрезмерному упрощению модели и её недообучению. И наоборот, большое значение C заставляет модель становиться жёстче, минимизируя любые ошибки классификации на тренировочных данных. Это приводит к точной подгонке под обучающие примеры, но сужает разделяющую полосу, вследствие чего модель становится чрезмерно чувствительной к шумам и выбросам, теряя способность к обобщению и сильно рискуя переобучиться.
2. Наилучшие результаты показало полиномиальное ядро четвертой степени.

Вывод:

В работе исследовалась эффективность SVM с разными ядрами на нелинейных данных. Эксперименты подтвердили, что полиномиальное ядро четвертой степени обеспечивает минимальную ошибку классификации и является оптимальным выбором для данной задачи.