Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и технологий
Высшая школа программной инженерии

Лабораторная работа №4

по дисциплине «Машинное обучение»

Выплолнил студент
гр. 33534/5 Стойкоски Н.С.

Руководитель И.А. Селин

Санкт-Петербург
2019 г.

Оглавление

Постановка задачи

1. Постройте алгоритм метода опорных векторов типа "*C-classification*" с параметром C = 1, используя ядро "*linear*" (LinearSVC или SVC с ядром “*linear*”). Визуализируйте разбиение пространства признаков на области с помощью полученной модели (пример визуализации). Выведите количество полученных опорных векторов, а также ошибки классификации на обучающей и тестовой выборках.

2. Используя алгоритм метода опорных векторов типа "*C-classification*" с линейным ядром (LinearSVC или SVC с ядром “*linear*”), добейтесь нулевой ошибки сначала на обучающей выборке, а затем на тестовой, путем изменения параметра C. Выберите оптимальное значение данного параметра и объясните свой выбор. Всегда ли нужно добиваться минимизации ошибки на обучающей выборке?

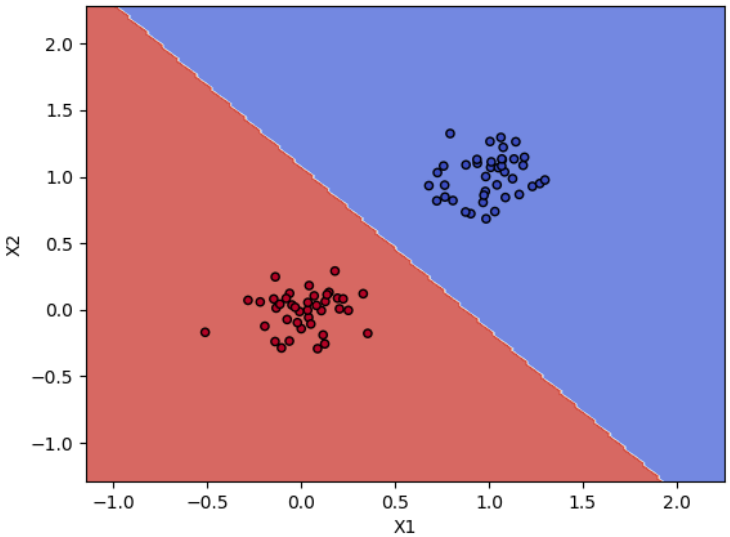
3. Среди ядер "*poly*", "*rbf*" и "*sigmoid*" выберите оптимальное в плане количества ошибок на тестовой выборке. Попробуйте различные значения параметра degree для полиномиального ядра.

4. Среди ядер "*poly*", "*rbf*" и "*sigmoid*" выберите оптимальное в плане количества ошибок на тестовой выборке.

5. Среди ядер "*poly* ", "*rbf* " и "*sigmoid*" выберите оптимальное в плане количества ошибок на тестовой выборке. Изменяя значение параметра gamma, продемонстрируйте эффект переобучения, выполните при этом визуализацию разбиения пространства признаков на области.

Ход работы

1. Был построен алгоритм метода опорных векторов типа "C-classification" с параметром C=1, с ядром "linear" на наборе svmdata1, svmdata1test. Было визуализировано разбиение пространства признаков на области с помощью полученной модели.



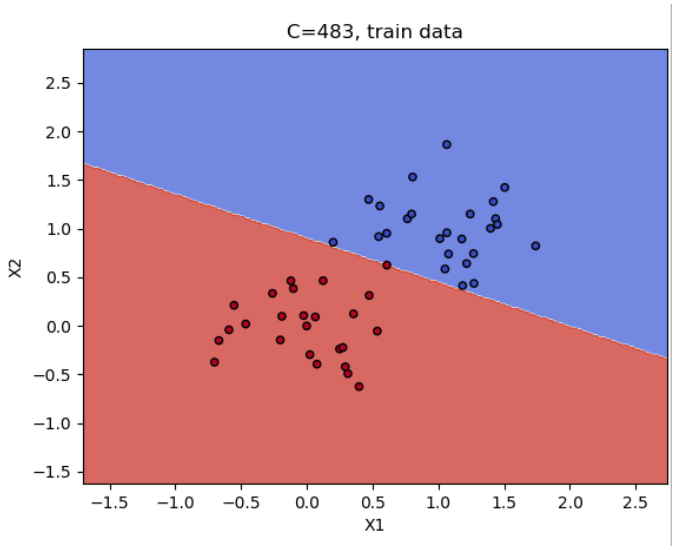
Количество опорных векторов: [3, 3]

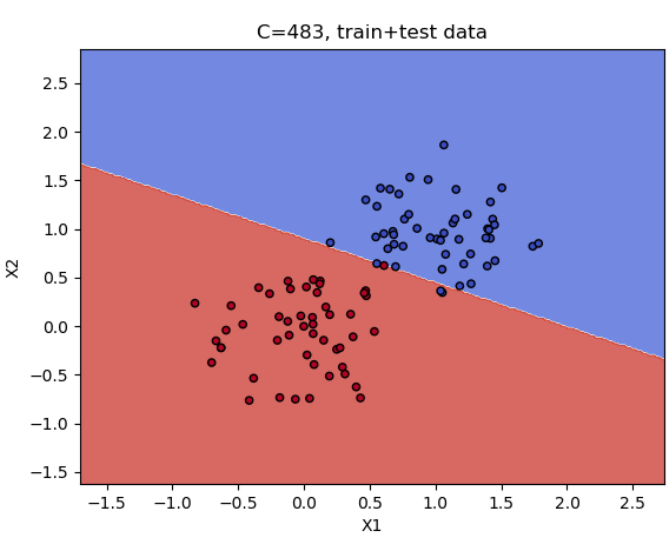
Точность на обучающая выборка: 1.0

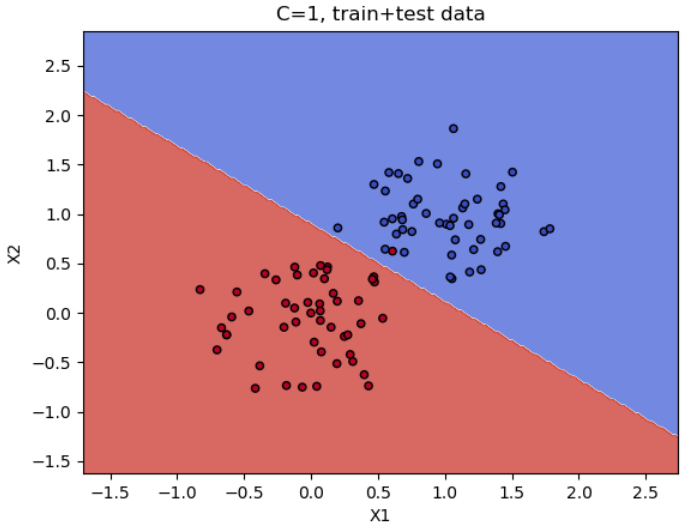
Точность на тестовая выборка: 1.0

2. Путем изменения параметра C была получена нулевая ошибка сначала на обучающей выборке (C=483) , а затем на тестовой (C=1). При этом, можно заметить что значение параметра C=483 приводит к переобучением и поэтому добиваемся меньшей точностью на тестовой выборке (C=483, TestAcc = 0,94) по сравнению с непереобученной моделью (C=1, TestAcc = 1.0). Можно сделать вывод что более оптимально выбрать параметр C=1 и не нужно всегда добиваться минимизации ошибки на обучающей выборке.

C = 483, TrainAcc = 1.0 TestAcc = 0.94
C = 1, TrainAcc = 0.98 TestAcc = 1.0

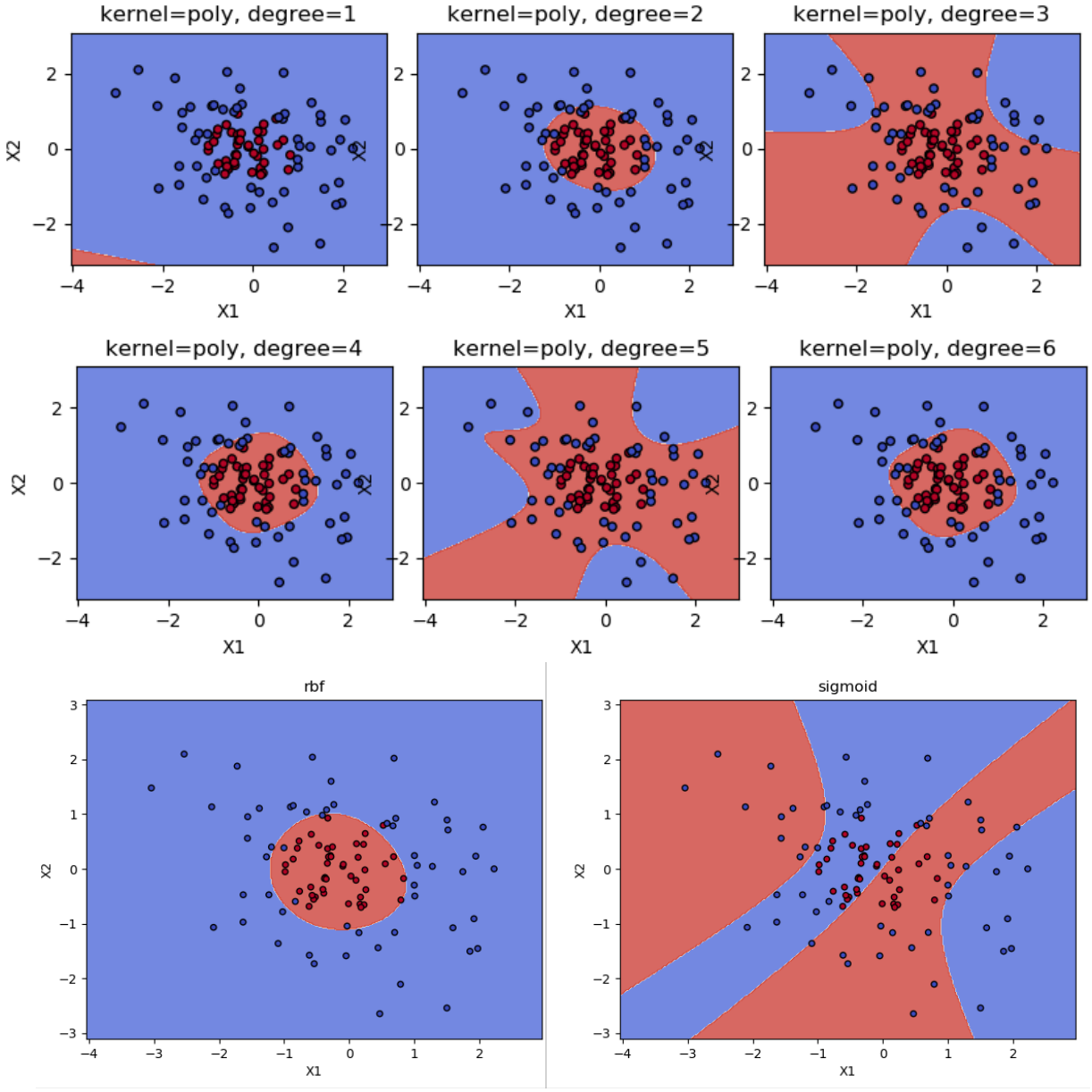






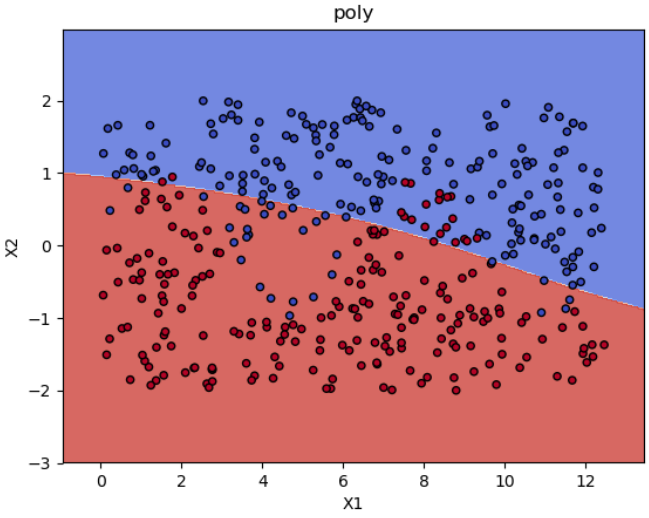
3. На наборе svmdata3, svmdata3test были построени алгоритмы метода опрных векторов с ядер "poly", "rbf" и "sigmoid". Были рассмотрени разные значения параметра degree для полиномиального ядра. Так же было визуализировано разбиение пространства признаков на области. Было найдено, что оптимальное в плане количества ошибок на тестовой выборке является ядро "rbf".

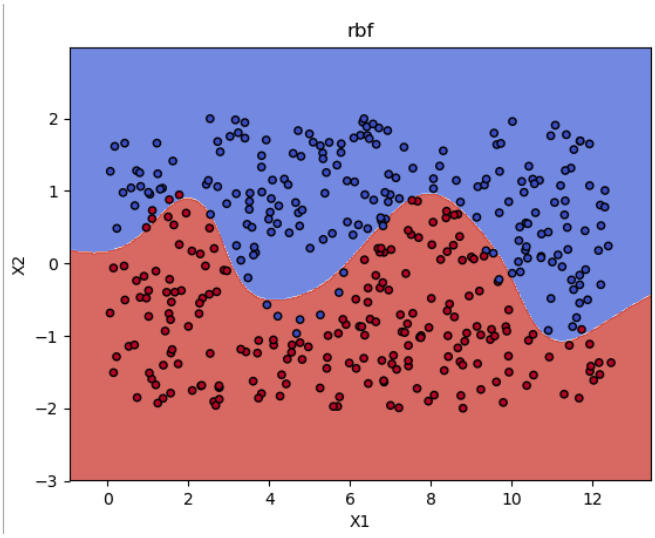
kernel=poly, degree=1, TestAcc = 0.58
kernel=poly, degree=2, TestAcc = 0.84
kernel=poly, degree=3, TestAcc = 0.54
kernel=poly, degree=4, TestAcc = 0.8
kernel=poly, degree=5, TestAcc = 0.46
kernel=poly, degree=6, TestAcc = 0.78
kernel=rbf, TestAcc = **0.94**
kernel=sigmoid, TestAcc = 0.46

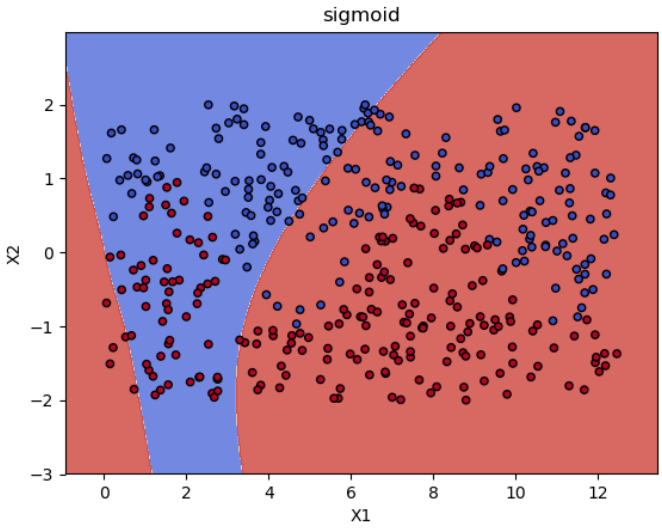


4. Аналогично как в (3) были использовани модели с ядрам "poly", "rbf" и "sigmoid" на наборе svmdata4, svmdata4test. Найдено что оптимальное в плане количества ошибок на тестовой выборке является ядро "rbf".

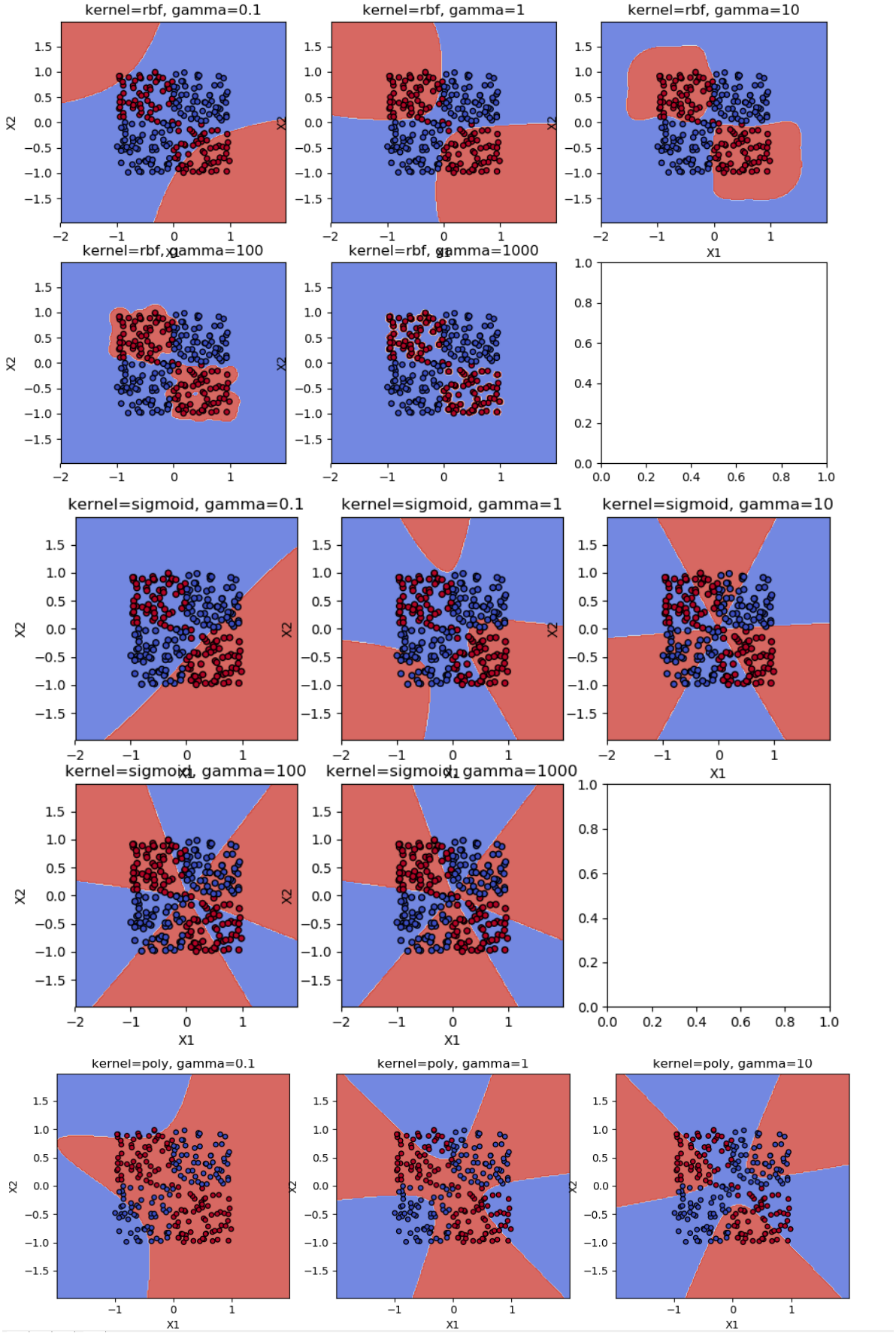
kernel=poly, TestAcc = 0.865
kernel=rbf, TestAcc = **0.935**
kernel=sigmoid, TestAcc = 0.485







5. На наборе svmdata5, svmdata5test были использованы модели с ядрам "poly", "rbf" и "sigmoid". При этом были рассмотрени разные значения параметра gamma. Найдено, что оптимальное в плане количества ошибок на тестовой выборке является ядро "rbf" с параметром gamma=10. Так же выполнена визуализация разбиения пространства признаков на области для соответствующие модели и продемонстрирован эффект переобучения.


kernel=rbf, gamma=0.1
kernel=rbf, gamma=1
kernel=rbf, gamma=10
kernel=rbf, gamma=100
kernel=rbf, gamma=1000
kernel=sigmoid, gamma=0.1
kernel=sigmoid, gamma=1
kernel=sigmoid, gamma=10
kernel=sigmoid, gamma=100
kernel=sigmoid, gamma=1000
kernel=poly, gamma=0.1
kernel=poly, gamma=1
kernel=poly, gamma=10
X1
X2

kernel=poly, gamma=0.1, TestAcc = 0.5333
kernel=poly, gamma=1, TestAcc = 0.51666
kernel=poly, gamma=10, TestAcc = 0.525
kernel=rbf, gamma=0.1, TestAcc = 0.625
kernel=rbf, gamma=1, TestAcc = 0.9166
kernel=rbf, gamma=10, TestAcc = **0.925**
kernel=rbf, gamma=100, TestAcc = 0.908
kernel=rbf, gamma=1000, TestAcc = 0.675
kernel=sigmoid, gamma=0.1, TestAcc = 0.6416
kernel=sigmoid, gamma=1, TestAcc = 0.5833
kernel=sigmoid, gamma=10, TestAcc = 0.5
kernel=sigmoid, gamma=100, TestAcc = 0.45
kernel=sigmoid, gamma=1000, TestAcc = 0.4583

Вывод

В ходе данной лабораторной работы был получен опыт работы с методом опорных векторов для решения задач классификации. Было исследовано как разные ядра и соответствующие параметры влияют на характер обученного моделя при визуализации разбиения пространства признаков на области.

Текст программы

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.svm import SVC
from sklearn import preprocessing, metrics

def make_meshgrid(x, y, h=.02):
    x_min, x_max = x.min() - 1, x.max() + 1
    y_min, y_max = y.min() - 1, y.max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                         np.arange(y_min, y_max, h))
    return xx, yy

def plot_contours(ax, clf, xx, yy, **params):
    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    out = ax.contourf(xx, yy, Z, **params)
    return out

def make_subplot(ax, clf, x, y, title=None):
    xx, yy = make_meshgrid(x[:, 0], x[:, 1])
    plot_contours(ax, clf, xx, yy, cmap=plt.cm.coolwarm, alpha=0.8)
    ax.scatter(x[:, 0], x[:, 1], c=y, cmap=plt.cm.coolwarm, s=20, edgecolors='k')
    ax.set_xlabel('X1')
```

```
    ax.set_ylabel('X2')
    ax.set_title(title)

def make_subplot2(ax, clf, x1, y1, x2, y2, title=None):
    make_subplot(ax, clf, x1, y1, title)
    ax.scatter(x2[:, 0], x2[:, 1], c=y2, cmap=plt.cm.coolwarm, s=20, edgecolors='k')

def make_plot1(clf, x, y, title=None):
    fig, sub = plt.subplots(1, 1)
    make_subplot(sub, clf, x, y, title)
    return sub

def make_plot2(clf, x1, y1, x2, y2, title=None):
    sub = make_plot1(clf, x1, y1, title)
    sub.scatter(x2[:, 0], x2[:, 1], c=y2, cmap=plt.cm.coolwarm, s=20, edgecolors='k')

def getData(filename, y_encoder):
    df1 = pd.read_csv(filename, delim_whitespace=True)
    x = df1[['X1', 'X2']].values
    y = y_encoder.fit_transform(df1['Colors'].values)
    return x, y

def task1():
    le = preprocessing.LabelEncoder()
    x_train, y_train = getData('svmdata1.txt', le)
    x_test, y_test = getData('svmdata1test.txt', le)
    clf = SVC(kernel='linear', C=1)
    clf.fit(x_train, y_train)
    make_plot2(clf, x_train, y_train, x_test, y_test)
    print('number of support vectors: ', clf.n_support_)
    print('train accuracy: ', metrics.accuracy_score(y_train, clf.predict(x_train)))
    print('test accuracy: ', metrics.accuracy_score(y_test, clf.predict(x_test)))

def task2():
    le = preprocessing.LabelEncoder()
    x_train, y_train = getData('svmdata2.txt', le)
    x_test, y_test = getData('svmdata2test.txt', le)

    clf = SVC(kernel='linear', C=483)
    clf.fit(x_train, y_train)
    train_accuracy = metrics.accuracy_score(y_train, clf.predict(x_train))
    test_accuracy = metrics.accuracy_score(y_test, clf.predict(x_test))
    print(f'C = 483, TrainAcc = {train_accuracy} TestAcc = {test_accuracy}')
    make_plot1(clf, x_train, y_train, 'C=483, train data')
    make_plot2(clf, x_train, y_train, x_test, y_test, 'C=483, train+test data')

    clf = SVC(kernel='linear', C=1)
    clf.fit(x_train, y_train)
    train_accuracy = metrics.accuracy_score(y_train, clf.predict(x_train))
    test_accuracy = metrics.accuracy_score(y_test, clf.predict(x_test))
    print(f'C = 1, TrainAcc = {train_accuracy} TestAcc = {test_accuracy}')
    make_plot2(clf, x_train, y_train, x_test, y_test, 'C=1, train+test data')

def task3():
    le = preprocessing.LabelEncoder()
    x_train, y_train = getData('svmdata3.txt', le)
    x_test, y_test = getData('svmdata3test.txt', le)

    fig, sub = plt.subplots(2, 3)
    plt.subplots_adjust(wspace=0.1, hspace=0.6)
```

```
    #poly
    for degree in range(1, 7):
        clf = SVC(kernel='poly', degree=degree, gamma='auto')
        clf.fit(x_train, y_train)
        test_accuracy = metrics.accuracy_score(y_test, clf.predict(x_test))
        print(f'kernel=poly, degree={degree}, TestAcc = {test_accuracy}')
        ax = sub.flatten()[degree-1]
        make_subplot2(ax, clf, x_train, y_train, x_test, y_test, f'kernel=poly,
degree={degree}')

    #rbf
    clf = SVC(kernel='rbf', gamma='auto')
    clf.fit(x_train, y_train)
    test_accuracy = metrics.accuracy_score(y_test, clf.predict(x_test))
    print(f'kernel=rbf, TestAcc = {test_accuracy}')
    make_plot2(clf, x_train, y_train, x_test, y_test, 'rbf')

    #sigmoid
    clf = SVC(kernel='sigmoid', gamma='auto')
    clf.fit(x_train, y_train)
    test_accuracy = metrics.accuracy_score(y_test, clf.predict(x_test))
    print(f'kernel=sigmoid, TestAcc = {test_accuracy}')
    make_plot2(clf, x_train, y_train, x_test, y_test, 'sigmoid')


def task4():
    le = preprocessing.LabelEncoder()
    x_train, y_train = getData('svmdata4.txt', le)
    x_test, y_test = getData('svmdata4test.txt', le)

    clf = SVC(kernel='poly', gamma='auto')
    clf.fit(x_train, y_train)
    test_accuracy = metrics.accuracy_score(y_test, clf.predict(x_test))
    print(f'kernel=poly, TestAcc = {test_accuracy}')
    make_plot2(clf, x_train, y_train, x_test, y_test, 'poly')

    clf = SVC(kernel='rbf', gamma='auto')
    clf.fit(x_train, y_train)
    test_accuracy = metrics.accuracy_score(y_test, clf.predict(x_test))
    print(f'kernel=rbf, TestAcc = {test_accuracy}')
    make_plot2(clf, x_train, y_train, x_test, y_test, 'rbf')

    clf = SVC(kernel='sigmoid', gamma='auto')
    clf.fit(x_train, y_train)
    test_accuracy = metrics.accuracy_score(y_test, clf.predict(x_test))
    print(f'kernel=sigmoid, TestAcc = {test_accuracy}')
    make_plot2(clf, x_train, y_train, x_test, y_test, 'sigmoid')

def task5():
    le = preprocessing.LabelEncoder()
    x_train, y_train = getData('svmdata5.txt', le)
    x_test, y_test = getData('svmdata5test.txt', le)

    gammas = [0.1, 1, 10, 100, 1000]

    fig, sub = plt.subplots(1, 3)
    for i in range(3):
        gamma = gammas[i]
        clf = SVC(kernel='poly', gamma=gamma)
        clf.fit(x_train, y_train)
        test_accuracy = metrics.accuracy_score(y_test, clf.predict(x_test))
```

```
        print(f'kernel=poly, gamma={gamma}, TestAcc = {test_accuracy}')
        ax = sub.flatten()[i]
        make_subplot2(ax, clf, x_train, y_train, x_test, y_test, f'kernel=poly,
gamma={gamma}')

    fig, sub = plt.subplots(2, 3)
    for i in range(5):
        gamma = gammas[i]
        clf = SVC(kernel='rbf', gamma=gamma)
        clf.fit(x_train, y_train)
        test_accuracy = metrics.accuracy_score(y_test, clf.predict(x_test))
        print(f'kernel=rbf, gamma={gamma}, TestAcc = {test_accuracy}')
        ax = sub.flatten()[i]
        make_subplot2(ax, clf, x_train, y_train, x_test, y_test, f'kernel=rbf,
gamma={gamma}')

    fig, sub = plt.subplots(2, 3)
    for i in range(5):
        gamma = gammas[i]
        clf = SVC(kernel='sigmoid', gamma=gamma)
        clf.fit(x_train, y_train)
        test_accuracy = metrics.accuracy_score(y_test, clf.predict(x_test))
        print(f'kernel=sigmoid, gamma={gamma}, TestAcc = {test_accuracy}')
        ax = sub.flatten()[i]
        make_subplot2(ax, clf, x_train, y_train, x_test, y_test, f'kernel=sigmoid,
gamma={gamma}')


task1()
task2()
task3()
task4()
task5()
plt.show()
```