Министерство науки и высшего образования Российской Федерации
федеральное государственное автономное образовательное учреждение
высшего образования
«Санкт-Петербургский политехнический университет Петра Великого»
(ФГАОУ ВО СПбПУ)
Институт промышленного менеджмента, экономики и торговли
Высшая инженерно-экономическая школа

ЛАБОРАТОРНАЯ РАБОТА №3

По дисциплине «Многомерный статистический анализ»

Построение SV-классификатора
(семестр 2)

Студент
группы
3740105/20101 _____
подпись, дата К.С. Малышева.

Оценка выполненной студентом работы:

Преподаватель,
Доцент, канд.физ-мат.наук _____
подпись, дата Л.В. Павлова

Санкт-Петербург – 2023

Задача работы:

Необходимо построить SV-классификатор на основе предварительно созданной последовательности векторов для разного типа данных: hard-margin, soft-margin, non-linear.

Дополнительно произвести исследование влияния результата классификации от штрафного параметра C и σ^2.

Ход работы:

1. Подготовить ТП объема ~ 100, состоящую из точек в двумерном пространстве, допускающих линейное разделение. Привести формулировку Вольфа соответствующей задачи квадратичного программирования и решить эту задачу (в Матлабе ф-я quadprog). Привести графическую интерпретацию исходных данных и полученных результатов.

2. Внести ошибки (несколько) в ТП. Построить SV-машину для «запорченных» данных. Исследовать влияние штрафного параметра C на ширину полосы (M). Вычислить и вывести M для каждого значения C и привести графическую интерпретацию результатов SV-обучения.

3. Подготовить ТП объема ~ 100, состоящую из двумерных векторов, не допускающих линейное разделение (при этом – разделимых нелинейно!). Повторить п.1), заменив скалярное произведение векторов гауссовым ядром. Проследить зависимость результатов от выбора C и σ^2.

4. Сделать выводы.

Постановка задачи

Исследование будет произведено на языке программирования Python в бесплатной интерактивной облачной среде для работы с кодом Google Colab.

Для выполнения работы необходимо импортировать следующие библиотеки.

```
import numpy as np
from numpy import linalg
from numpy.linalg import norm
import cvxopt
import cvxopt.solvers
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.metrics import accuracy_score
```

В каждом рассматриваемом случае необходимо сгенерировать набор данных для обучения, которые предварительно бы были разделены на два класса со значениями -1 и 1.

Постановка задачи обучения:

$$\boldsymbol{x} \in R^n \sim P(\boldsymbol{x});\ \boldsymbol{x} \to y \sim P(y|\boldsymbol{x})$$

$$F = \{ f(\boldsymbol{x}): R^n \to R \}$$

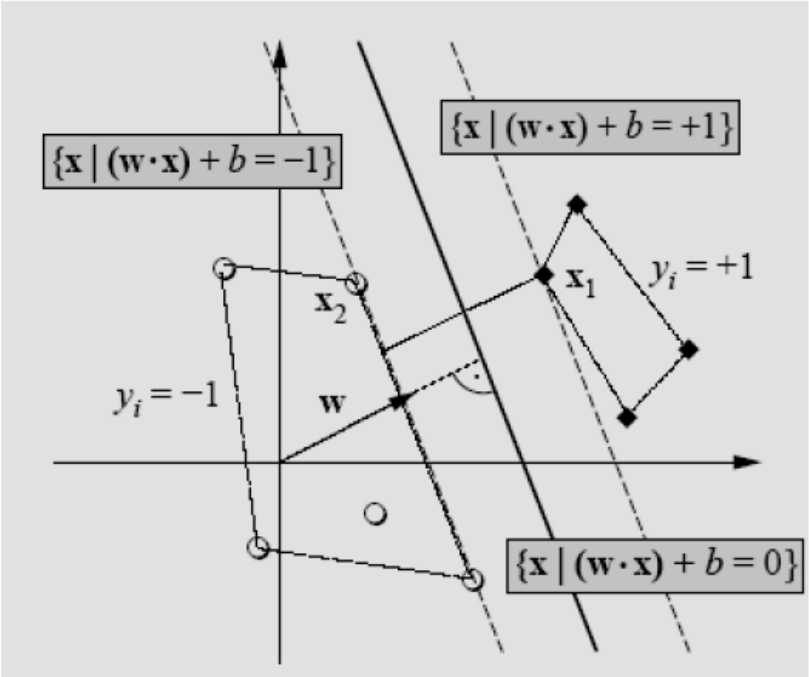
$$\text{ТП: } (\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_l, y_l),\ P(\boldsymbol{x}, y) = P(\boldsymbol{x}) \times P(y|\boldsymbol{x}),$$

$$\boldsymbol{x}_i \in R^n,\ y_i \in \{\pm 1\},\ i = 1, \ldots, l$$

$$F = \{ f(\boldsymbol{x}): R^n \to \{\pm 1\} \}$$

По смоделированным данным необходимо найти такую гиперплоскость, которая наилучшим образом разделяла данные на две группы, а также смогла бы предсказать класс подаваемой тестовой выборки.

Для проверки точности классификации разделим наши данные на две группы: обучающая выборка, по которой будет строиться SV-классификатор, и тестовая выборка, на которой будет тестироваться классификатор.



$$f(x) = (w \cdot x) + b \qquad (5)$$

$$w \cdot x_i + b \geq 1 \quad \text{для } y_i = +1$$
$$w \cdot x_i + b \leq -1 \quad \text{для } y_i = -1$$

$$y_i((w \cdot x_i) + b) - 1 \geq 0, \ \forall i \qquad (6)$$

Представим уравнение прямой как *x*w+b=0* и масштабируем параметры так, чтобы ближайшие к прямой точки данных удовлетворяли *x*w+b=±1* (плюс или минус в зависимости от класса) — эти точки и называют *опорными векторами*.

В таком случае расстояние между граничными точками классов равно 2/|*w*|.

$$\left.\begin{aligned} w \cdot x_1 + b &= +1 \\ w \cdot x_2 + b &= -1 \end{aligned}\right\} \Rightarrow (w \cdot (x_1 - x_2)) = 2 = \|w\| \times \mathbf{M}$$

$$\Rightarrow \mathbf{M} = 2/\|w\| \qquad (7)$$

Очевидно, мы хотим максимизировать эту величину, чтобы как можно более качественно отделить классы. Поэтому задача оптимизации сводиться к минимизации $\frac{1}{2}|w|^2$.

$$\left.\begin{aligned} \boldsymbol{w}\cdot\boldsymbol{x}_1+b=+1 \\ \boldsymbol{w}\cdot\boldsymbol{x}_2+b=-1 \end{aligned}\right\} \quad \Rightarrow (\boldsymbol{w}\cdot(\boldsymbol{x}_1-\boldsymbol{x}_2))=2=\|\boldsymbol{w}\|\times\mathbf{M}$$

$$\Rightarrow \mathbf{M}=2/\|\boldsymbol{w}\| \qquad (7)$$

$$\textit{Исходная задача:} \quad \min_{w}\frac{1}{2}\|\boldsymbol{w}\|^2$$

$$y_i\left((\boldsymbol{w}\cdot\boldsymbol{x}_i)+b\right)-1\geq 0, \quad i=1,\ldots,l \qquad (8)$$

Если ее решить, то классификация по запросу будет производиться следующим образом.

$$\textbf{SV-машина}: f(\boldsymbol{x})=sgn\left((\boldsymbol{w}\cdot\boldsymbol{x})+b\right)$$

Такой вид задачи подходит только для разделения сепарабельного случая. Чтобы в дальнейшем разделить другие виды данных, рекомендуется сразу перейти к двойственной задаче квадратичного программирования.

Для случая hard-margin задача будет иметь следующий вид.

Двойственная задача в формулировке Wolf'a:

$$\max_{\alpha} L_D, \qquad L_D(\boldsymbol{\alpha})=\sum_{i=1}^{l}\alpha_i-\frac{1}{2}\sum_{i,j=1}^{l}\alpha_i\alpha_j y_i y_j\left(\boldsymbol{x}_i\cdot\boldsymbol{x}_j\right) \qquad (14)$$

$$\sum_{i=1}^{l}\alpha_i y_i=0, \quad \alpha_i\geq 0$$

В случае soft-margin добавляется также штрафной параметр C.

Двойственная задача в формулировке Wolf'a:

$$\max_{\alpha} L_D(\boldsymbol{\alpha}),$$

$$L_D(\boldsymbol{\alpha})=\sum_{i=1}^{l}\alpha_i-\frac{1}{2}\sum_{i,j=1}^{l}\alpha_i\alpha_j y_i y_j(\boldsymbol{x}_i\cdot\boldsymbol{x}_j)$$

$$0\leq\alpha_i\leq C, i=1,\ldots,l, \quad \sum_{i=1}^{l}\alpha_i y_i=0 \qquad (23)$$

Введем функцию $W(\alpha) = -L_D(\alpha)$ и перейдем к записи задачи в матричном виде:

$$\min_{\alpha} W(\alpha), \quad W(\alpha) = \frac{1}{2}\alpha^T Q\,\alpha - \alpha^T \cdot \mathbf{1}$$

при ограничениях:

$$\alpha^T y = 0$$

$$\mathbf{0} \le \alpha \le C \cdot \mathbf{1}$$

(здесь обычное умножение константы C на единичный вектор)

$\alpha, y, 0, 1 \in R^l$ (т.е. это l – мерные векторы, l – объем ТП)

Элементы матицы Q: $Q_{ij} = y_i y_j k(x_i, x_j)$, $i, j = 1, \ldots, l$

Таким образом, мы найдем опорный вектора - те, для которых значения коэффициентов Лагранжа $\alpha_i \ge tol$, tol=1e-3. Также при введении штрафного параметра C появляются BSV векторы, значения коэффициентов Лагранжа которых равно данному параметру.

$$\boldsymbol{x}_i: \quad 0<\alpha_i<C, \quad \xi_i=0 \quad \text{(USVs)}$$

$$\boldsymbol{x}_i: \quad \alpha_i=C, \quad \xi_i>0 \quad \text{(BSVs)}$$

А SVM будет выглядеть следующим образом.

$$\textbf{SV-машина}: f(\boldsymbol{x}) = sgn((\boldsymbol{w}\cdot\boldsymbol{x}) + b)$$

$$\downarrow$$

$$f(\boldsymbol{x}) = sgn\left(\sum_{i=1}^{l} \alpha_i y_i (\boldsymbol{x}_i \cdot \boldsymbol{x}) + b\right) \qquad (20)$$

Для данных, которые разделены нелинейно, необходимо использовать *ядра (ядерные функции*).

$$(\boldsymbol{x}_i \cdot \boldsymbol{x}) \rightarrow k(\boldsymbol{x}_i, \boldsymbol{x})$$

$$k(\boldsymbol{x}, \boldsymbol{x}') = (\varphi(\boldsymbol{x}) \cdot \varphi(\boldsymbol{x}'))$$

Ядерная матрица будет иметь следующий вид.

$$K = \begin{bmatrix} K(x_1,x_1) & K(x_1,x_2) \cdots \\ K(x_2,x_1) & \\ \vdots & \end{bmatrix}$$

Таким образом, для нелинейного разделения SVM будет выглядеть следующим образом.

$$f(\boldsymbol{x}) = \operatorname{sgn}\left(\sum_{i=1}^{l} \alpha_i y_i k(\boldsymbol{x}_i, \boldsymbol{x}) + b\right)$$

По итогу для реализации исследования с помощью программных средств изначально сделаем следующие допущения:

1) данные будут сразу преобразованы и представлены в виде ядра $k(\boldsymbol{x}, \boldsymbol{x}')$, в т.ч. для линейно разделимых векторов;

2) задача оптимизации сразу будет представлена двойственной задачей квадратичного программирования.

1. Hard-margin SVM – классификация (случай линейной разделимости данных/сепарабельный случай)

Для начала сгенерируем данные в двумерном пространстве объемом 200, допускающих линейное разделение. 100 значением присваивается метка 1, другим 100 – метка -1.

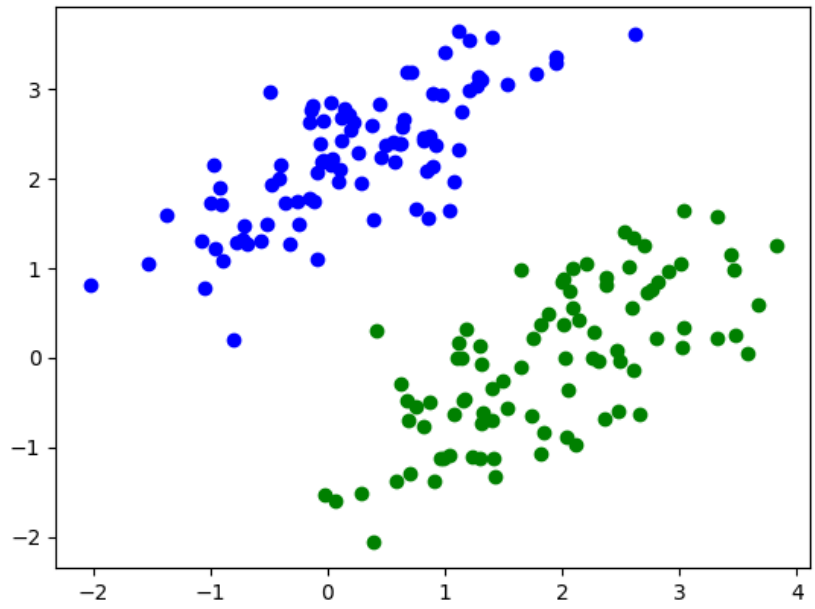
```
#Генерация линейно разделенных данных
mean1 = np.array([0, 2])
mean2 = np.array([2, 0])
cov = np.array([[0.8, 0.6], [0.6, 0.8]])
X1 = np.random.multivariate_normal(mean1, cov, 100)
y1 = np.ones(len(X1))
X2 = np.random.multivariate_normal(mean2, cov, 100)
y2 = np.ones(len(X2)) * -1
```

Для того, чтобы в дальнейшем проверить точность классификации разделим данные на обучающую и тестовую выборки в соотношении 180/20 соответственно.

```
# Разделение на обучающую и тестовую последовательность
X1_train = X1[:90]
y1_train = y1[:90]
X2_train = X2[:90]
y2_train = y2[:90]
X_train = np.vstack((X1_train, X2_train))
y_train = np.hstack((y1_train, y2_train))

X1_test = X1[90:]
y1_test = y1[90:]
X2_test = X2[90:]
y2_test = y2[90:]
X_test = np.vstack((X1_test, X2_test))
y_test = np.hstack((y1_test, y2_test))
```

Отобразим данные обучающей выборки на графике.

```
plt.plot(X1_train[:,0], X1_train[:,1],"bo")
plt.plot(X2_train[:,0], X2_train[:,1], "go")
plt.axis("tight")
plt.show()
```



Перейдем к созданию SV-классификатора.

Линейное ядро имеет следующий вид, где метод np.dot является скалярным умножением двух векторов.

```
def linear_kernel(x1, x2):
    return np.dot(x1, x2)
```

Для удобства дальнейшей оптимизации преобразуем данные в матрицу K.

Поставим и решим задачу оптимизации с помощь библиотеки cvxopt и метода qp.

```
# Ставим задачу квадратичного программирования

P = cvxopt.matrix(np.outer(y_train,y_train) * K)
q = cvxopt.matrix(np.ones(n_samples) * -1)
A = cvxopt.matrix(y_train, (1,n_samples))
b = cvxopt.matrix(0.0)

G = cvxopt.matrix(np.diag(np.ones(n_samples) * -1))
h = cvxopt.matrix(np.zeros(n_samples))

# Решаем задачу квадратичного программирования
solution = cvxopt.solvers.qp(P, q, G, h, A, b)
solution
```

Из получившегося решения выведем коэффициенты Лагранжа и представим в виде одномерного массива через метод np.ravel.

```
# Коэф Лагранжа
a = np.ravel(solution['x'])
a
```

```
array([1.23543870e-09, 7.37323788e-10, 8.80719998e-10, 1.71452138e-09,
       8.24490614e-10, 9.66408445e-10, 8.01507281e-10, 7.70755022e-10,
       4.40752623e-09, 9.37229690e-10, 6.89091536e-10, 1.31203291e-09,
       1.80369863e-09, 7.97772380e-10, 2.45026855e+00, 1.16699139e-09,
       1.11301093e-09, 1.07803641e-09, 7.19538191e-10, 9.49623836e-10,
       8.81475662e-10, 1.23507748e-09, 1.18534356e-09, 1.11238292e-09,
       6.31626271e-10, 7.27504655e-10, 7.63395429e-10, 7.60908772e-10,
       7.65716415e-10, 8.82260051e-10, 3.25603197e-07, 9.65414836e-10,
       1.71413082e-09, 8.36761186e-10, 7.59578270e-10, 7.82776616e-10,
       1.79534724e-09, 1.91755108e+00, 1.10441849e-09, 8.08631275e-10,
       8.13116200e-10, 9.61938541e-10, 8.84359495e-10, 9.56691576e-10,
       7.69276543e-10, 6.95502634e-10, 8.12547496e-10, 8.46092531e-10,
       1.66836414e-09, 7.30433138e-10, 7.21442208e-10, 7.39019398e-10,
       1.17288522e-09, 1.00929841e-09, 1.02840893e-09, 9.15627130e-10,
       1.20033709e-09, 5.21951526e-09, 1.58837159e-09, 9.83614641e-10,
       7.39393286e-10, 8.65359860e-10, 8.97451305e-10, 1.07888610e-09,
       1.21675250e-09, 1.03846733e-09, 1.08329265e-09, 6.84161825e-10,
       1.31266966e-09, 7.12265617e-09, 1.06218038e-09, 7.46042423e-10,
       1.12837495e-09, 9.16835812e-10, 9.02783283e-10, 7.04004780e-10,
       7.32424338e-10, 1.01478497e-09, 1.22291184e-09, 7.82222421e-10,
       9.56858711e-10, 8.52734320e-10, 3.22955215e-09, 8.39999836e-10,
       6.81389540e-10, 7.30973718e-10, 1.13574917e-09, 1.04915272e-09,
       1.05059885e-09, 1.04096040e-09, 1.06932970e-09, 1.01290323e-09,
       6.66495233e-10, 6.61245107e-10, 6.31447977e-10, 9.97706341e-10,
       1.46588675e-09, 1.03759324e-09, 1.16413567e-09, 6.43858509e-10,
       1.05337252e-09, 6.83954068e-10, 7.92492590e-10, 8.29512952e-10,
       9.42881350e-10, 1.01177410e-09, 8.47764753e-10, 1.19472182e-09,
       8.40554153e-10, 7.62497775e-10, 7.86107699e-10, 1.04667309e-09,
       6.14120378e-10, 8.51837701e-10, 9.11003383e-10, 7.62866566e-10,
       7.45988936e-10, 1.12811329e-09, 8.11044396e-10, 9.06170101e-10,
       6.73423482e-10, 8.55601483e-10, 1.20277886e-09, 7.04568178e-10,
       8.11742523e-10, 7.05509862e-10, 7.50615757e-10, 6.49889178e-10,
       1.70548867e-09, 9.14332195e-10, 8.83302073e-10, 9.91389659e-10,
       6.66290026e-10, 7.66934679e-10, 1.00594791e-09, 8.46025493e-10,
       7.18914281e-10, 1.38244874e-09, 6.49319044e-10, 6.55831151e-10,
       7.22748764e-10, 7.68456503e-10, 8.42836355e-10, 7.21859690e-10,
       8.55896375e-10, 7.13004539e-10, 1.29421881e-09, 8.04806958e-10,
       7.65793449e-10, 9.99564948e-10, 7.73141538e-10, 7.00183671e-10,
       7.56893336e-10, 4.36781997e+00, 7.00027419e-10, 6.83126444e-10,
       7.86558378e-10, 6.49300419e-10, 1.12611700e-09, 6.46047826e-10,
       6.37204954e-10, 1.38949859e-09, 8.90638347e-10, 1.15130929e-09,
       8.81183115e-10, 7.53385154e-10, 1.08051242e-09, 7.08021751e-10,
       7.39815613e-10, 9.83066653e-10, 6.79938372e-10, 7.22066473e-10,
       6.86505261e-10, 1.26407281e-09, 1.51052612e-09, 1.20688111e-09,
       6.16057033e-10, 7.86043352e-10, 1.12721254e-09, 1.29615842e-09])
```

Найдем опорные вектора, у которых $\alpha_i \geq tol,\ tol$=1e-3. .

```
sv = a > 1e-3
ind = np.arange(len(a))[sv]
a = a [sv]

sv_x = X_train[ind]
sv_y = y_train[ind]

print("%d support vectors out of %d points" % (len(a), n_samples))
```

```
3 support vectors out of 180 points
```

По итогу нашлось 3 опорных вектора со следующими значениями.

```
a
```

```
array([2.45026855, 1.91755108, 4.36781997])
```

```
sv_x
```

```
array([[-0.80273663,  0.1913919 ],
       [ 1.03732717,  1.63405499],
       [ 0.42259604,  0.29222615]])
```

```
sv_y
```

```
array([ 1.,  1., -1.])
```

Рассчитаем вектор w.

$$\boldsymbol{w} = \sum_{i=1}^{l} \alpha_i y_i \boldsymbol{x}_i$$

```
# Весовой вектор w
w = np.zeros(n_features)
for n in range(len(a)):
  w += a[n] * sv_y[n] * sv_x[n]
w
```

```
array([-1.82361592,  2.32595424])
```

А также ширину полосы Margin.

```
margin = 2/norm(w)
round(margin,2)
```

```
0.68
```

Отдельно рассчитаем параметр b.

$$(w \cdot x) + b = 0 \rightarrow \sum_{i=1}^{l} \alpha_i y_i (x_i \cdot x) + b = 0$$

```
# Коэф b
b = 0
for n in range(len(a)):
  b += sv_y[n]
  b -= np.sum(a * sv_y * K[ind[n],sv])
b /= len(a)
b
```

```
-0.9090515681686856
```

Создадим итоговую SV-машину и классифицируем для примера тестовую выборку.

```
#Классификация тестовой выборки
f = np.dot(X_test, w) + b
y_predict = np.sign(f)
```

Рассчитаем точность классификатора. В данном случае она составляет 100%.

Точность классификации

```
[18] print('Точность классификации:', accuracy_score(y_test, y_predict))
     correct = np.sum(y_predict == y_test)
     print("%d out of %d predictions correct" % (correct, len(y_predict)))
```

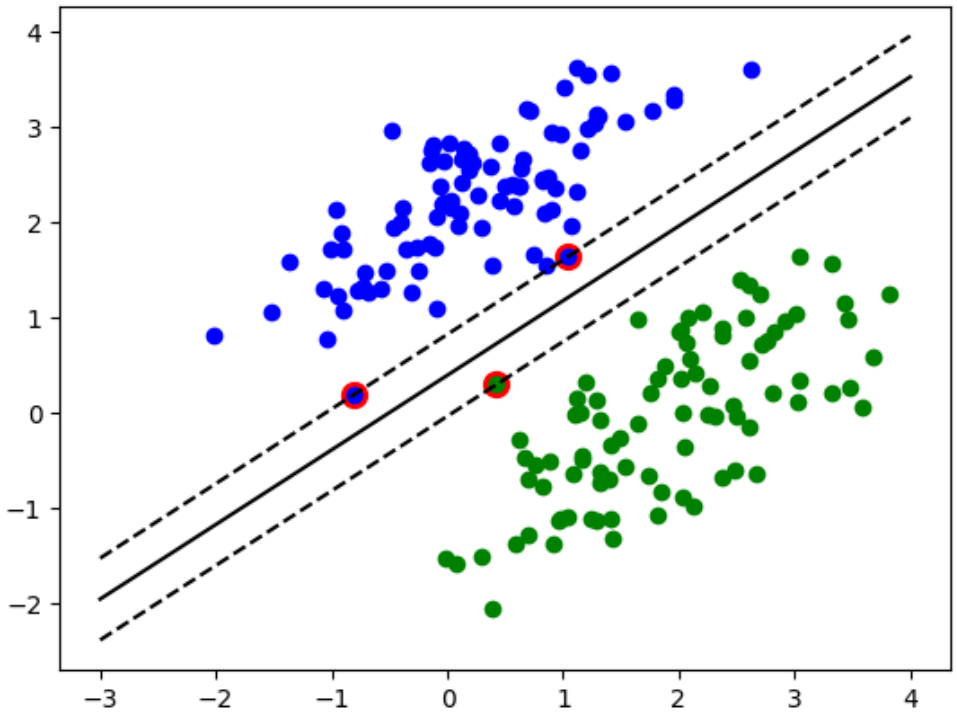
```
Точность классификации: 1.0
20 out of 20 predictions correct
```

20 из 20 значений были верно классифицированы.

По обучающей выборке построим график, на котором отобразим опорные векторы.

```
def f(x, w, b, c=0):
  # w.x + b = c
  return (-w[0] * x - b + c) / w[1]
```

```
plt.plot(X1_train[:,0], X1_train[:,1],"bo")
plt.plot(X2_train[:,0], X2_train[:,1], "go")
plt.scatter(sv_x[:,0], sv_x[:,1], s=100, c="r")

# w.x + b = 0
a0 = -3; a1 = f(a0, w, b)
b0 = 4; b1 = f(b0, w, b)
plt.plot([a0,b0], [a1,b1], "k")

# w.x + b = 1
a0 = -3; a1 = f(a0, w, b, 1)
b0 = 4; b1 = f(b0, w, b, 1)
plt.plot([a0,b0], [a1,b1], "k--")

# w.x + b = -1
a0 = -3; a1 = f(a0, w, b, -1)
b0 = 4; b1 = f(b0, w, b, -1)
plt.plot([a0,b0], [a1,b1], "k--")

plt.axis("tight")
plt.show()
```



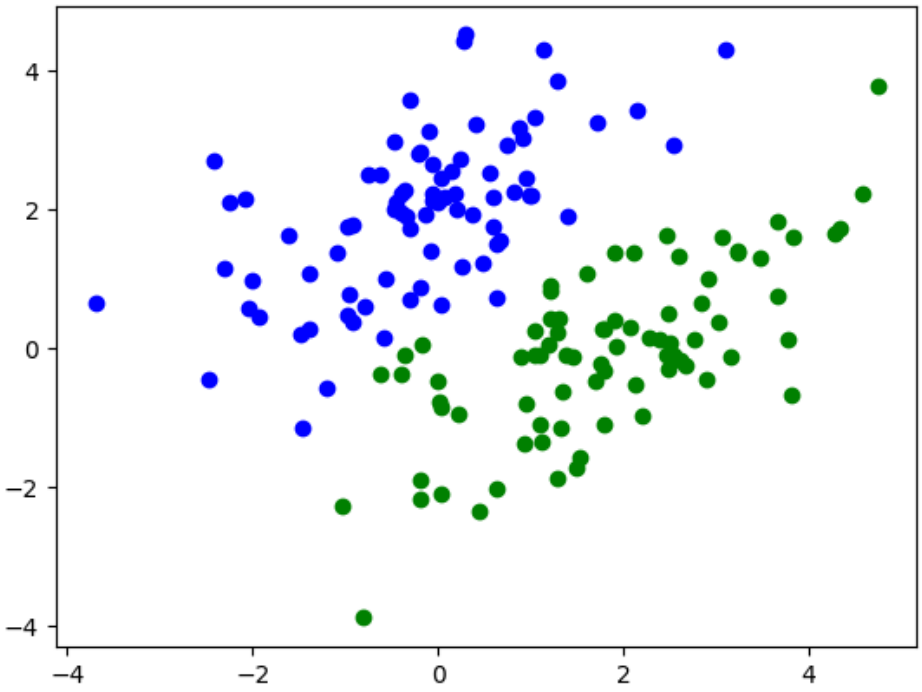
2. Soft-margin SVM – классификация (случай линейной неразделимости данных/несепарабельный случай)

Изменив параметры среднего и ковариации, сгенерируем линейно неразделимые данные.

```
#Генерация "запорченных" данных
mean1 = np.array([0, 2])
mean2 = np.array([2, 0])
cov = np.array([[1.5, 1.0], [1.0, 1.5]])
X1 = np.random.multivariate_normal(mean1, cov, 100)
y1 = np.ones(len(X1))
X2 = np.random.multivariate_normal(mean2, cov, 100)
y2 = np.ones(len(X2)) * -1
```

Аналогично прошлому случаю разделим данные на обучающую и тестовую выборку в соотношении 160/40, и построим график.

```
plt.plot(X1_train[:,0], X1_train[:,1],"bo")
plt.plot(X2_train[:,0], X2_train[:,1], "go")
plt.axis("tight")
plt.show()
```



Также создаем линейное ядро и матрицу K.

```
def linear_kernel(x1, x2):
    return np.dot(x1, x2)
```

```
kernel = linear_kernel
# Разделение на кол-во наблюдений и на кол-во признаков
n_samples, n_features = X_train.shape


K = np.zeros((n_samples, n_samples))
for i in range(n_samples):
  for j in range(n_samples):
    K[i,j] = kernel(X_train[i], X_train[j])
```

В данном случае двойственная задача будет отличаться на штрафной параметр C. Для начала установим его размер 1.

```
# Ставим задачу квадратичного программирования
C = 1

P = cvxopt.matrix(np.outer(y_train,y_train) * K)
q = cvxopt.matrix(np.ones(n_samples) * -1)
A = cvxopt.matrix(y_train, (1,n_samples))
b = cvxopt.matrix(0.0)

tmp1 = np.diag(np.ones(n_samples) * -1)
tmp2 = np.identity(n_samples)
G = cvxopt.matrix(np.vstack((tmp1, tmp2)))
tmp1 = np.zeros(n_samples)
tmp2 = np.ones(n_samples) * C
h = cvxopt.matrix(np.hstack((tmp1, tmp2)))

# Решаем задачу квадратичного программирования
solution = cvxopt.solvers.qp(P, q, G, h, A, b)
solution
```

Также выделяем коэффициент Лагранжа из решения и опорные векторы.

```
# Коэф Лагранжа
a = np.ravel(solution['x'])
a
```

В данном случае при параметре C=1 общее количество опорных векторов составило 15, из них USV = 6, а BSV = 9.

```
# Опорные векторы имеют ненулевые множители Лагранжа
sv = a > 1e-3
bsv = a==C
ind = np.arange(len(a))[sv]
indb = np.arange (len(a))[bsv]
a = a [sv]

sv_x = X_train[ind]
sv_y = y_train[ind]
bsv_x = X_train[indb]
bsv_y = y_train[indb]

print("%d SV out of %d points" % (len(a), n_samples))
print("%d USV out of %d points" % (len(a) - len(bsv_x), n_samples))
print("%d BSV out of %d points" % (len(bsv_x), n_samples))
```

```
15 SV out of 160 points
6 USV out of 160 points
9 BSV out of 160 points
```

Считаем параметры w и b.

```
# Весовой вектор w
w = np.zeros(n_features)
for n in range(len(a)):
  w += a[n] * sv_y[n] * sv_x[n]
w
```

```
array([-1.69232155,  2.0012502 ])
```

```
b = 0
for n in range(len(a)):
  b += sv_y[n]
  b -= np.sum(a * sv_y * K[ind[n],sv])
b /= len(a)
b
```

```
-0.4919846487689437
```

В данном случае ширина полосы составила 0,76.

```
margin = 2/norm(w)
round(margin,2)
```

```
0.76
```

Также проведем классификацию тестовой выборки и оценим ее точность.

```
[203] #Классификация тестовой выборки
      f = np.dot(X_test, w) + b
      y_predict = np.sign(f)
```

Точность классификации

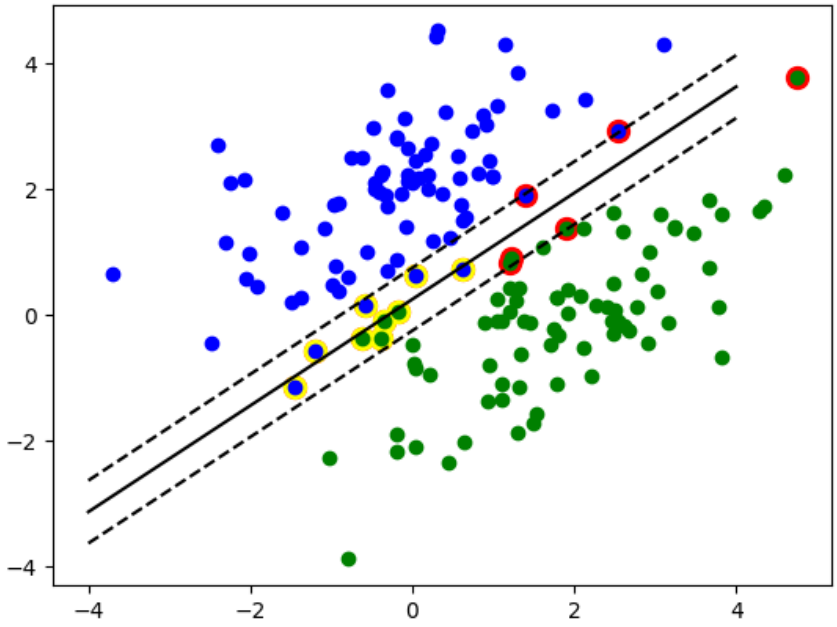
```
[204] print('Точность классификации:', accuracy_score(y_test, y_predict))
      correct = np.sum(y_predict == y_test)
      print("%d out of %d predictions correct" % (correct, len(y_predict)))
```

```
Точность классификации: 1.0
40 out of 40 predictions correct
```

Построим график.

```
def f(x, w, b, c=0):
  # w.x + b = c
  return (-w[0] * x - b + c) / w[1]
```

```
plt.plot(X1_train[:,0], X1_train[:,1],"bo")
plt.plot(X2_train[:,0], X2_train[:,1], "go")
plt.scatter(sv_x[:,0], sv_x[:,1], s=100, c="r")
plt.scatter(bsv_x[:,0], bsv_x[:,1], s=100, c="yellow")

# w.x + b = 0
a0 = -4; a1 = f(a0, w, b)
b0 = 4; b1 = f(b0, w, b)
plt.plot([a0,b0], [a1,b1], "k")

# w.x + b = 1
a0 = -4; a1 = f(a0, w, b, 1)
b0 = 4; b1 = f(b0, w, b, 1)
plt.plot([a0,b0], [a1,b1], "k--")

# w.x + b = -1
a0 = -4; a1 = f(a0, w, b, -1)
b0 = 4; b1 = f(b0, w, b, -1)
plt.plot([a0,b0], [a1,b1], "k--")

plt.axis("tight")
plt.show()
```



Проведем аналогичные действия, изменяя штрафной параметр C.

При штрафном параметре C=0.1:

```
# Опорные векторы имеют ненулевые множители Лагранжа
sv = a > 1e-3
bsv = a==C
ind = np.arange(len(a))[sv]
indb = np.arange (len(a))[bsv]
a = a [sv]

sv_x = X_train[ind]
sv_y = y_train[ind]
bsv_x = X_train[indb]
bsv_y = y_train[indb]

print("%d SV out of %d points" % (len(a), n_samples))
print("%d USV out of %d points" % (len(a) - len(bsv_x), n_samples))
print("%d BSV out of %d points" % (len(bsv_x), n_samples))
```

```
33 SV out of 160 points
9 USV out of 160 points
24 BSV out of 160 points
```

Количество опорных векторов увеличилось до 33 векторов, из которых USV = 9, BSV = 24.

```
margin = 2/norm(w)
round(margin,2)
```

```
1.51
```

Ширина полосы увеличилась до 1,51.

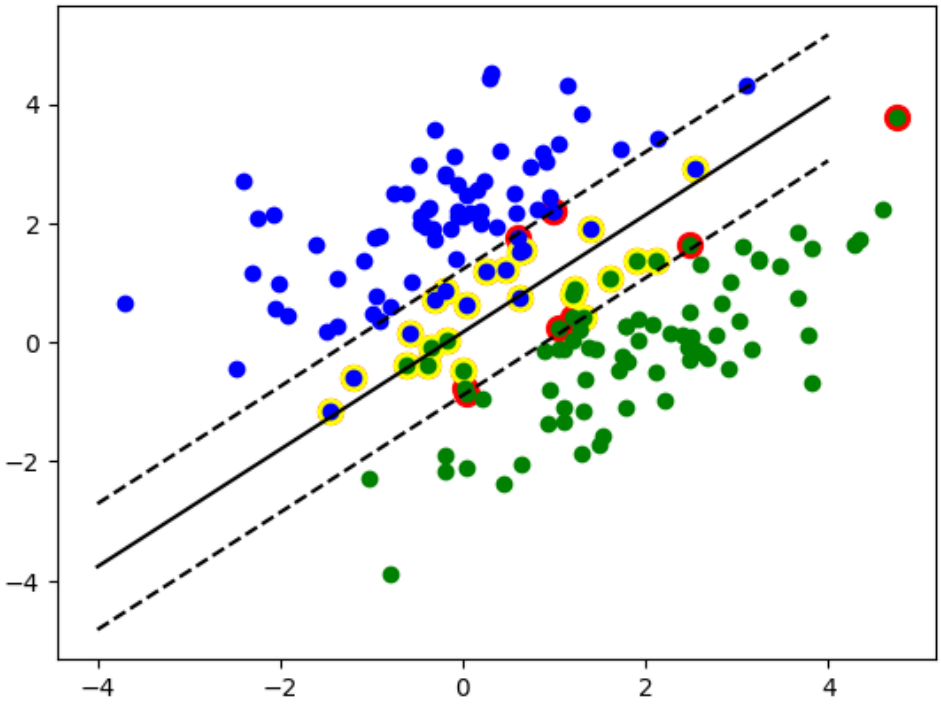
Точность классификации

```
[221] print('Точность классификации:', accuracy_score(y_test, y_predict))
      correct = np.sum(y_predict == y_test)
      print("%d out of %d predictions correct" % (correct, len(y_predict)))
```

```
Точность классификации: 1.0
40 out of 40 predictions correct
```

Точность классификации составила 100%.

График классификации выглядит следующим образом.



При штрафном параметре C=10:

```
# Опорные векторы имеют ненулевые множители Лагранжа
sv = a > 1e-3
bsv = a==C
ind = np.arange(len(a))[sv]
indb = np.arange (len(a))[bsv]
a = a [sv]

sv_x = X_train[ind]
sv_y = y_train[ind]
bsv_x = X_train[indb]
bsv_y = y_train[indb]

print("%d SV out of %d points" % (len(a), n_samples))
print("%d USV out of %d points" % (len(a) - len(bsv_x), n_samples))
print("%d BSV out of %d points" % (len(bsv_x), n_samples))
```

```
9 SV out of 160 points
4 USV out of 160 points
5 BSV out of 160 points
```

Количество опорных векторов уменьшилось до 9 векторов, BSV = 5, USV = 4.

```
margin = 2/norm(w)
round(margin,2)
```

```
0.46
```

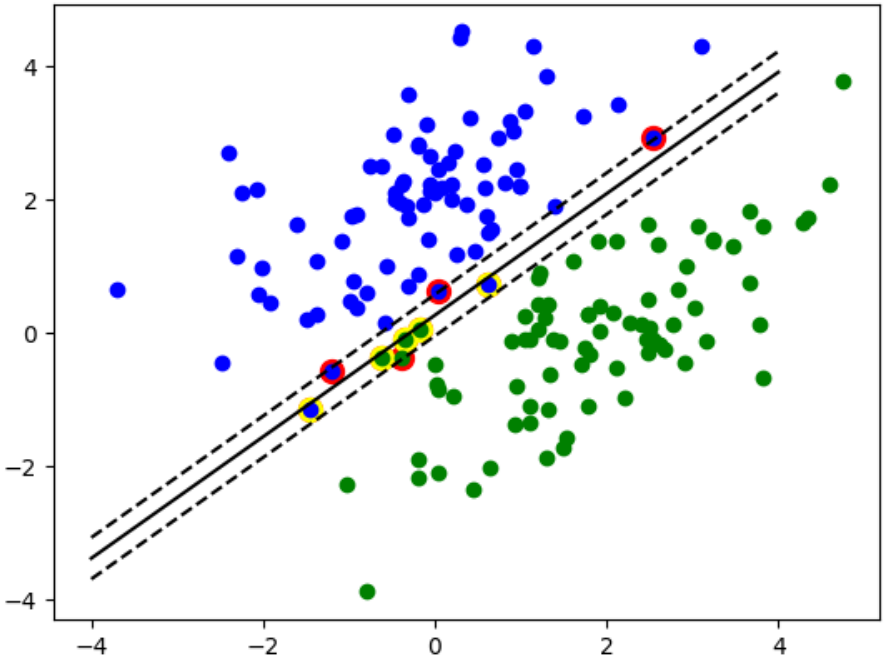
Ширина полосы уменьшилась до 0,46.

```
print('Точность классификации:', accuracy_score(y_test, y_predict))
correct = np.sum(y_predict == y_test)
print("%d out of %d predictions correct" % (correct, len(y_predict)))
```

```
Точность классификации: 0.975
39 out of 40 predictions correct
```

Точность классификации 97,5%.

График выглядит следующим образом.



При штрафном параметре C=100:

```
# Опорные векторы имеют ненулевые множители Лагранжа
sv = a > 1e-3
bsv = a==C
ind = np.arange(len(a))[sv]
indb = np.arange (len(a))[bsv]
a = a [sv]

sv_x = X_train[ind]
sv_y = y_train[ind]
bsv_x = X_train[indb]
bsv_y = y_train[indb]

print("%d SV out of %d points" % (len(a), n_samples))
print("%d USV out of %d points" % (len(a) - len(bsv_x), n_samples))
print("%d BSV out of %d points" % (len(bsv_x), n_samples))
```

```
7 SV out of 160 points
4 USV out of 160 points
3 BSV out of 160 points
```

Количество опорных векторов уменьшилось до 7 векторов, BSV = 4, USV = 3.

```
margin = 2/norm(w)
round(margin,2)
```

```
0.25
```

Ширина полосы уменьшилась до 0,25.

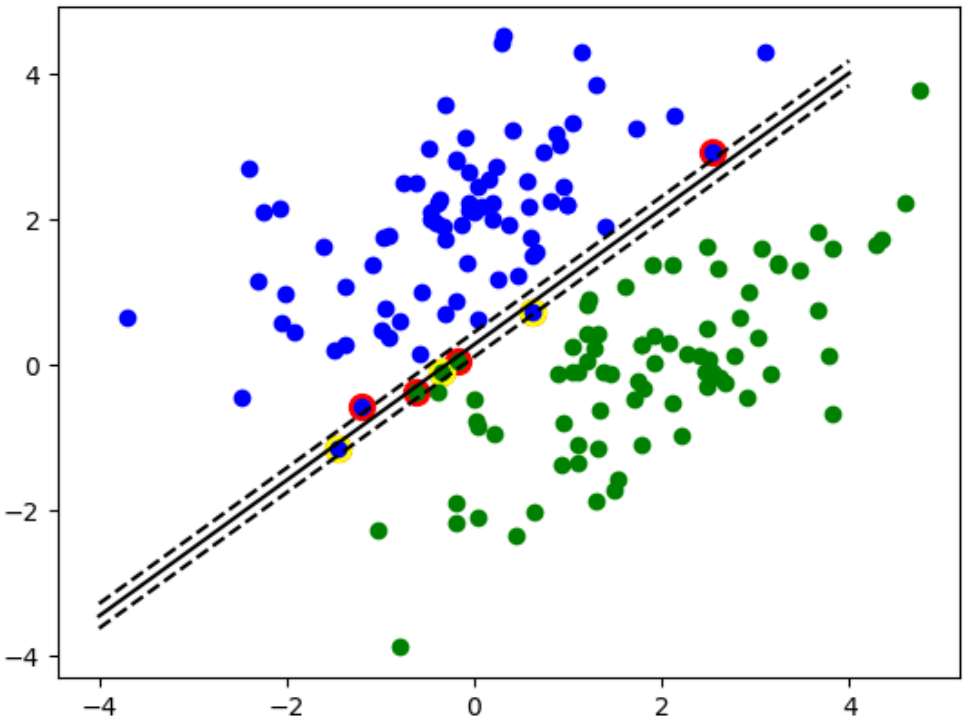
Точность классификации

```
[267] print('Точность классификации:', accuracy_score(y_test, y_predict))
      correct = np.sum(y_predict == y_test)
      print("%d out of %d predictions correct" % (correct, len(y_predict)))
```

```
Точность классификации: 0.975
39 out of 40 predictions correct
```

Точность классификации составила 97,5%

График классификации выглядит следующим образом.



При штрафном параметре C=1000:

```
# Опорные векторы имеют ненулевые множители Лагранжа
sv = a > 1e-3
bsv = a==C
ind = np.arange(len(a))[sv]
indb = np.arange (len(a))[bsv]
a = a [sv]

sv_x = X_train[ind]
sv_y = y_train[ind]
bsv_x = X_train[indb]
bsv_y = y_train[indb]

print("%d SV out of %d points" % (len(a), n_samples))
print("%d USV out of %d points" % (len(a) - len(bsv_x), n_samples))
print("%d BSV out of %d points" % (len(bsv_x), n_samples))
```

```
7 SV out of 160 points
3 USV out of 160 points
4 BSV out of 160 points
```

Количество векторов составило 7 векторов, BSV = 4, USV = 3.

```
margin = 2/norm(w)
round(margin,2)
```

```
0.25
```

Ширина полосы составила 0,25.

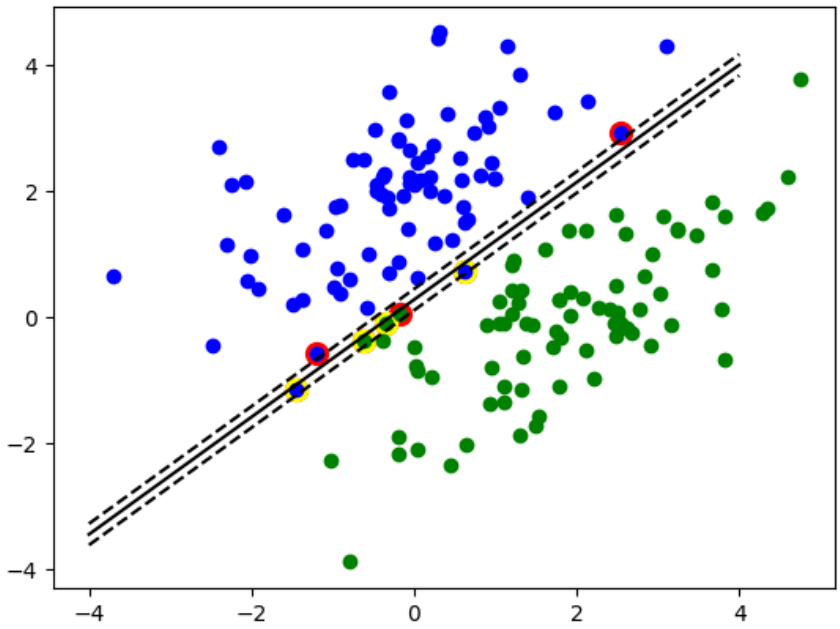
Точность классификации

```
[267] print('Точность классификации:', accuracy_score(y_test, y_predict))
      correct = np.sum(y_predict == y_test)
      print("%d out of %d predictions correct" % (correct, len(y_predict)))
```

```
Точность классификации: 0.975
39 out of 40 predictions correct
```

Точность классификации составила 97,5%.

График выглядит следующим образом.



При штрафном параметре C=10000 значения не отличаются от C=1000.

```
# Опорные векторы имеют ненулевые множители Лагранжа
sv = a > 1e-3
bsv = a==C
ind = np.arange(len(a))[sv]
indb = np.arange (len(a))[bsv]
a = a [sv]

sv_x = X_train[ind]
sv_y = y_train[ind]
bsv_x = X_train[indb]
bsv_y = y_train[indb]

print("%d SV out of %d points" % (len(a), n_samples))
print("%d USV out of %d points" % (len(a) - len(bsv_x), n_samples))
print("%d BSV out of %d points" % (len(bsv_x), n_samples))
```

```
7 SV out of 160 points
3 USV out of 160 points
4 BSV out of 160 points
```

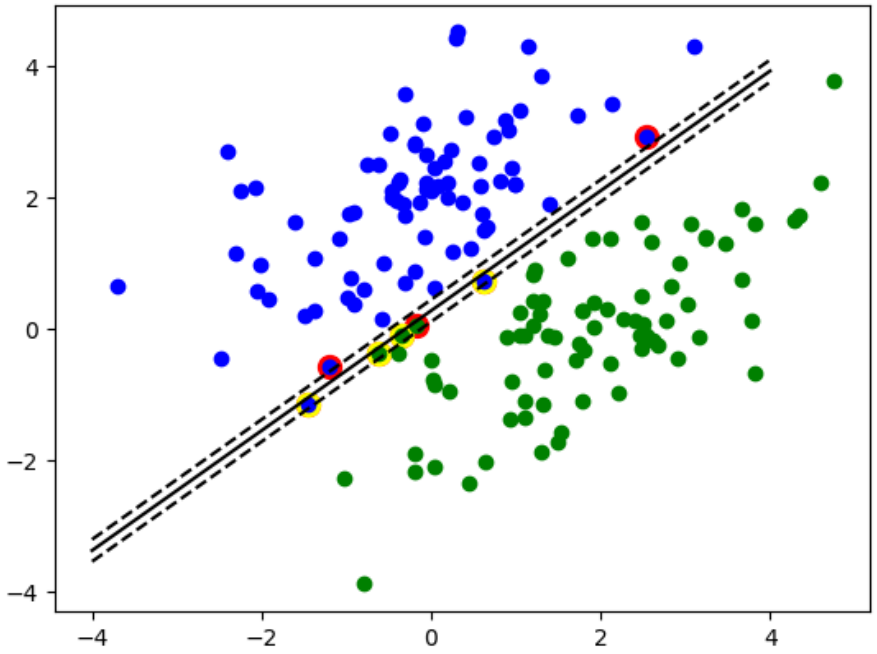
```
margin = 2/norm(w)
round(margin,2)
```

```
0.18
```

Точность классификации

```
[267] print('Точность классификации:', accuracy_score(y_test, y_predict))
      correct = np.sum(y_predict == y_test)
      print("%d out of %d predictions correct" % (correct, len(y_predict)))
```

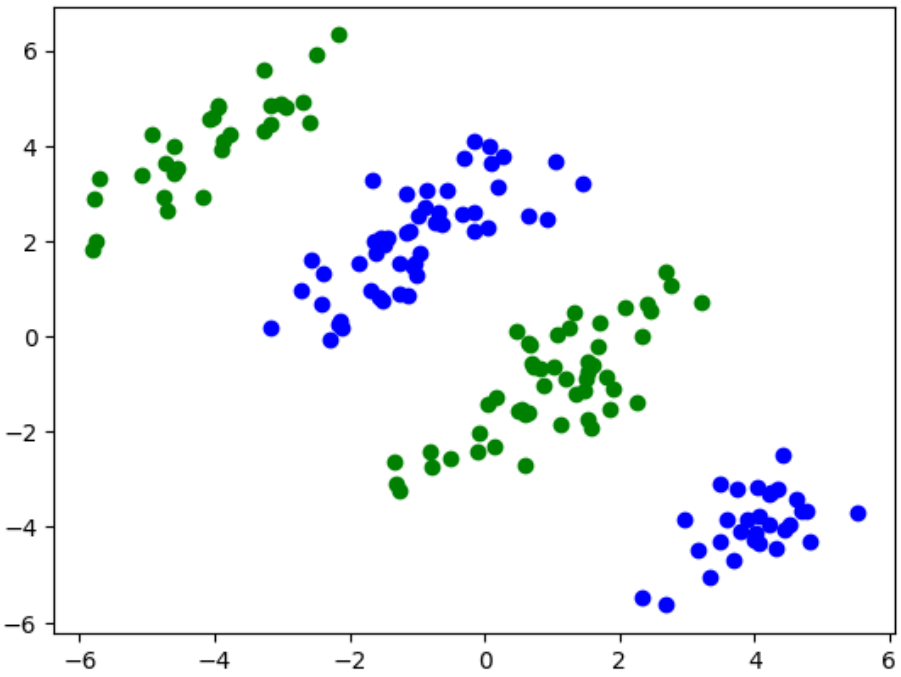
```
Точность классификации: 0.975
39 out of 40 predictions correct
```



3. Non-linear SVM – классификация (нелинейное обучение)

Сгенерируем данные, которые не разделяются линейно из двумерного распределения с разными параметрами.

```
#Генерация данных
mean1 = [-1, 2]
mean2 = [1, -1]
mean3 = [4, -4]
mean4 = [-4, 4]
cov = [[1.0,0.8], [0.8, 1.0]]
X1 = np.random.multivariate_normal(mean1, cov, 50)
X1 = np.vstack((X1, np.random.multivariate_normal(mean3, cov, 50)))
y1 = np.ones(len(X1))
X2 = np.random.multivariate_normal(mean2, cov, 50)
X2 = np.vstack((X2, np.random.multivariate_normal(mean4, cov, 50)))
y2 = np.ones(len(X2)) * -1
```



Разделяем на обучающую и тестовую выборку.

```
# Разделение на обучающую и тестовую последовательность
X1_train = X1[:80]
y1_train = y1[:80]
X2_train = X2[:80]
y2_train = y2[:80]
X_train = np.vstack((X1_train, X2_train))
y_train = np.hstack((y1_train, y2_train))

X1_test = X1[80:]
y1_test = y1[80:]
X2_test = X2[80:]
y2_test = y2[80:]
X_test = np.vstack((X1_test, X2_test))
y_test = np.hstack((y1_test, y2_test))
```

В этом случае заменяем линейное ядро на гауссовское.

$$k(x, x') = e^{-\|x-x'\|^2 / 2\sigma^2}$$

Для начала возьмем параметр σ=1 и создадим матрицу K.

```python
def gaussian_kernel(x, y, sigma=1):
    return np.exp(-linalg.norm(x-y)**2 / (2 * (sigma ** 2)))
```

```python
kernel = gaussian_kernel
# Разделение на кол-во наблюдений и на кол-во признаков
n_samples, n_features = X_train.shape

K = np.zeros((n_samples, n_samples))
for i in range(n_samples):
  for j in range(n_samples):
    K[i,j] = kernel(X_train[i], X_train[j])
```

В качестве штрафного параметра возьмем значение C=1 и решим задачу.

```python
# Ставим задачу квадратичного программирования
C = 1

P = cvxopt.matrix(np.outer(y_train,y_train) * K)
q = cvxopt.matrix(np.ones(n_samples) * -1)
A = cvxopt.matrix(y_train, (1,n_samples))
b = cvxopt.matrix(0.0)

tmp1 = np.diag(np.ones(n_samples) * -1)
tmp2 = np.identity(n_samples)
G = cvxopt.matrix(np.vstack((tmp1, tmp2)))
tmp1 = np.zeros(n_samples)
tmp2 = np.ones(n_samples) * C
h = cvxopt.matrix(np.hstack((tmp1, tmp2)))

# Решаем задачу квадратичного программирования
solution = cvxopt.solvers.qp(P, q, G, h, A, b)
solution
```

```
     pcost       dcost       gap    pres   dres
 0: -1.4132e+01 -2.4557e+02  2e+02  8e-15  5e-16
 1: -1.6998e+01 -3.0567e+01  1e+01  8e-15  4e-16
 2: -1.8188e+01 -2.0564e+01  2e+00  7e-15  2e-16
 3: -1.8553e+01 -1.9134e+01  6e-01  6e-15  2e-16
 4: -1.8643e+01 -1.8858e+01  2e-01  3e-15  2e-16
 5: -1.8682e+01 -1.8753e+01  7e-02  2e-15  2e-16
 6: -1.8695e+01 -1.8706e+01  1e-02  4e-15  2e-16
 7: -1.8697e+01 -1.8698e+01  1e-03  2e-16  2e-16
 8: -1.8697e+01 -1.8697e+01  3e-05  4e-16  2e-16
 9: -1.8697e+01 -1.8697e+01  5e-07  2e-16  2e-16
Optimal solution found.
{'x': <180x1 matrix, tc='d'>,
 'y': <1x1 matrix, tc='d'>,
 's': <360x1 matrix, tc='d'>,
 'z': <360x1 matrix, tc='d'>,
 'status': 'optimal',
 'gap': 5.293868843471202e-07,
 'relative gap': 2.831380089659597e-08,
 'primal objective': -18.697132408343162,
 'dual objective': -18.697132937730046,
 'primal infeasibility': 2.220446049250313e-16,
 'dual infeasibility': 1.958486850064716e-16,
 'primal slack': 7.481865100692007e-09,
 'dual slack': 9.823019818550958e-10,
 'iterations': 9}
```

Из решения выделим коэффициенты Лагранжа.

```python
# Коэф Лагранжа
a = np.ravel(solution['x'])
a
```

Количество опорных векторов составило 33 вектора, USV=32, BSV=1.

```
33 SV out of 160 points
32 USV out of 160 points
1 BSV out of 160 points
```

Рассчитываем функцию.

```
y_predict = np.zeros(len(X_train))
for i in range(len(X_train)):
  s = 0
  for a, sv_y in zip(a, sv_y):
    s += a * sv_y * kernel(X_train[i], sv_x)
  y_predict[i] = s

f = y_predict + b
```

```
y_predict = np.sign(f)
```

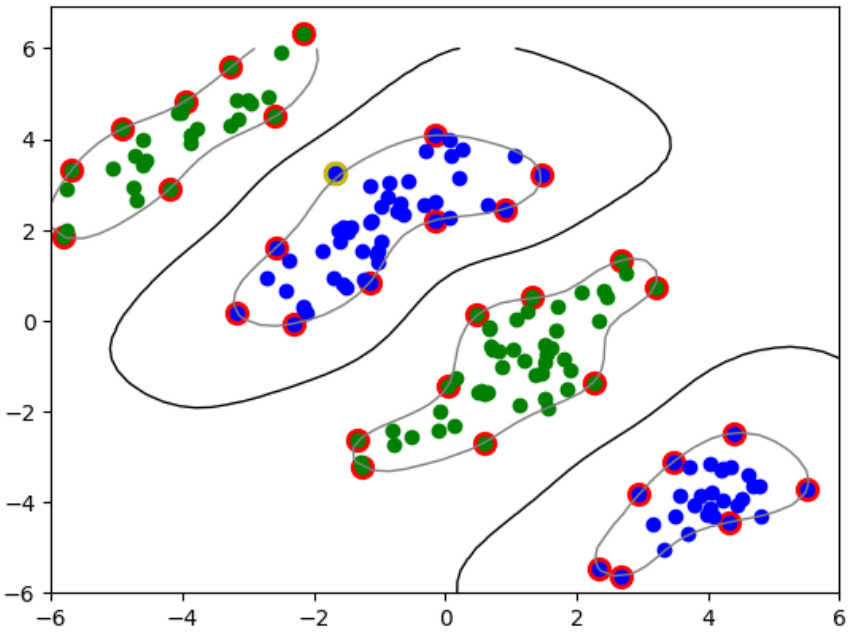
Рассчитаем также точность классификации на X_test.

Точность классификации

```
[13] correct = np.sum(clf.predict(X_test) == y_test)
     print('Точность классификации:', accuracy_score(y_test, clf.predict(X_test)))
     print("%d векторов из %d предсазаны верно" % (correct, len(clf.predict(X_test))))

     Точность классификации: 1.0
     40 векторов из 40 предсазаны верно
```

Построим график по полученным данным.

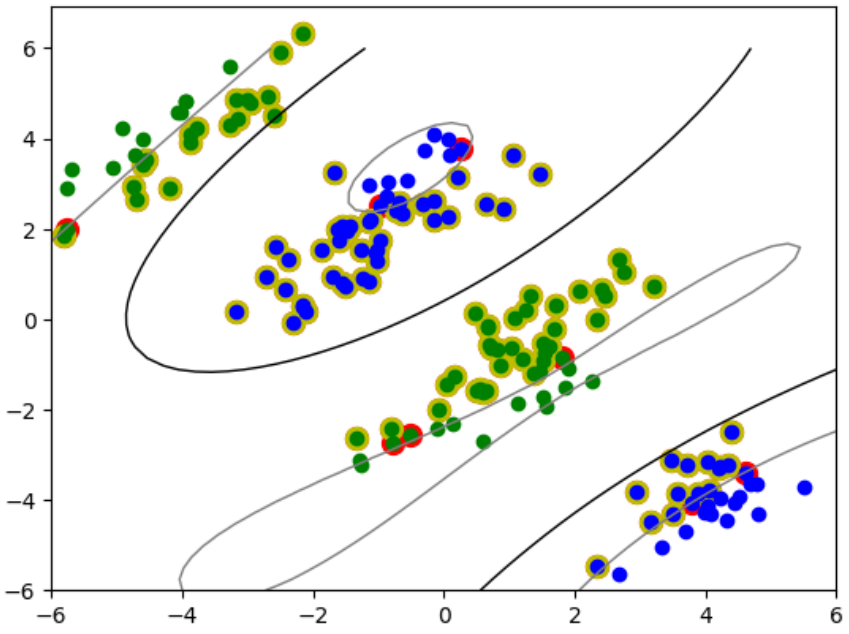
```
#Построение графика
plt.plot(X1_train[:,0], X1_train[:,1], "bo")
plt.plot(X2_train[:,0], X2_train[:,1], "go")
plt.scatter(clf.sv_x[:,0], clf.sv_x[:,1], s=100, c="r")
plt.scatter(clf.bsv_x[:,0], clf.bsv_x[:,1], s=100, c="y")


X1, X2 = np.meshgrid(np.linspace(-6,6,50), np.linspace(-6,6,50))
X = np.array([[x1, x2] for x1, x2 in zip(np.ravel(X1), np.ravel(X2))])
Z = clf.project(X).reshape(X1.shape)
plt.contour(X1, X2, Z, [0.0], colors='k', linewidths=1, origin='lower')
plt.contour(X1, X2, Z + 1, [0.0], colors='grey', linewidths=1, origin='lower')
plt.contour(X1, X2, Z - 1, [0.0], colors='grey', linewidths=1, origin='lower')

plt.axis("tight")
plt.show()
```



Рассмотрим различные вариации параметра σ и C.

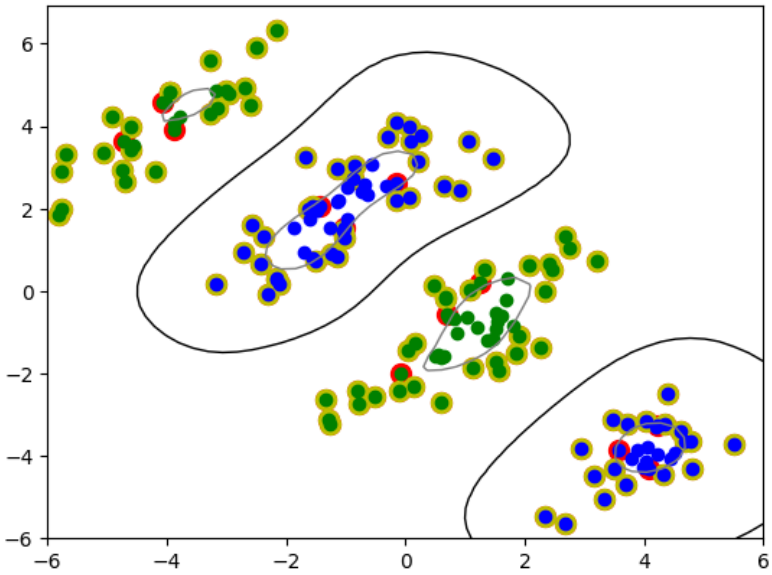
При C=1 и σ = 5:

Количество опорных векторов составит 115 векторов (USV=8,BSV=107), точность классификации останется на уровне 97,5%, а график будет выглядеть следующим образом.



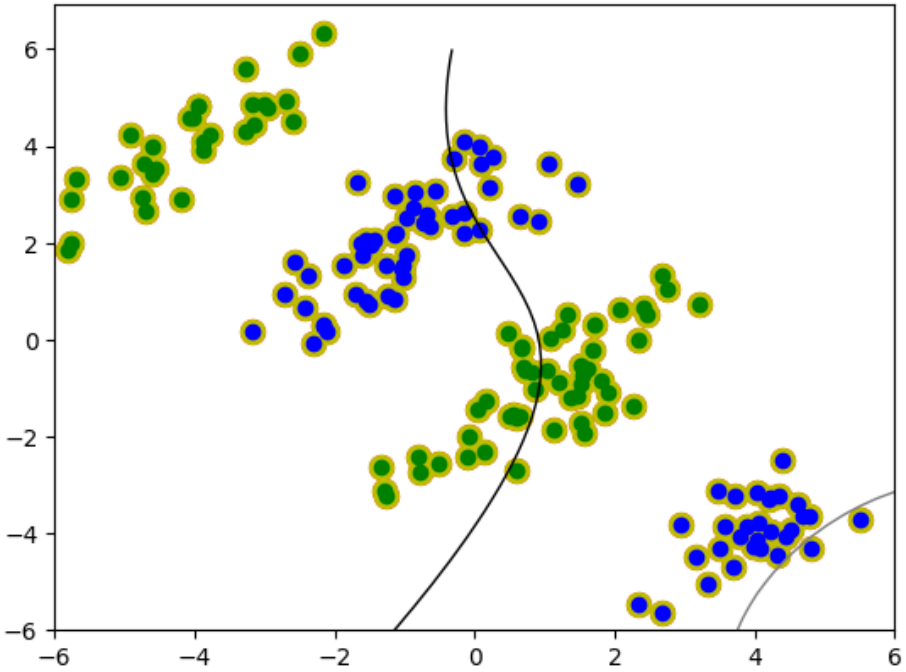
При C=0.1. и σ = 1:

Количество опорных векторов составит 113 векторов (USV=12, BSV=101), точность классификации останется на уровне 97,5%, а график будет выглядеть следующим образом.



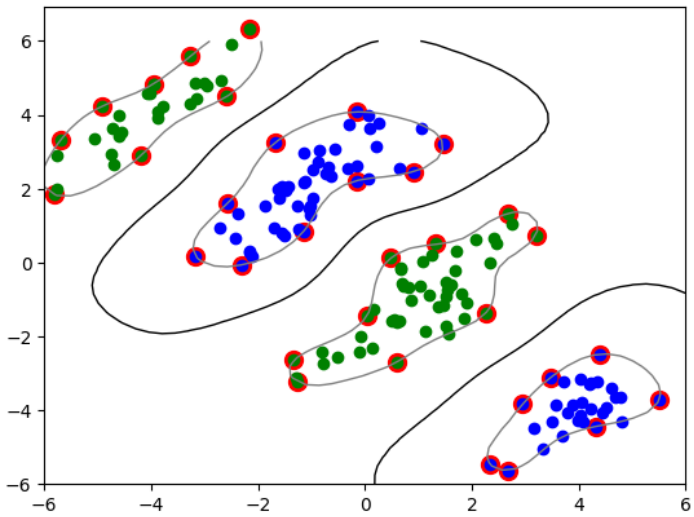
При C=0.1. и σ = 5:

Количество опорных векторов составит 160 векторов (USV=0, BSV=160), точность классификации останется на уровне 100%, а график будет выглядеть следующим образом.



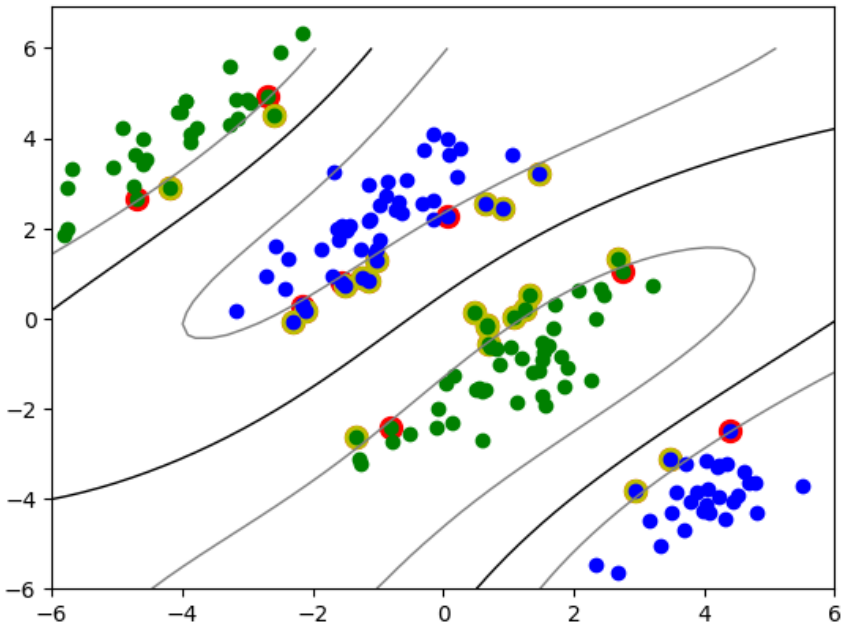
При C=10 и σ = 1:

Количество опорных векторов составит 33 вектора (USV=33, BSV=0), точность классификации останется на уровне 100%, а график будет выглядеть следующим образом.



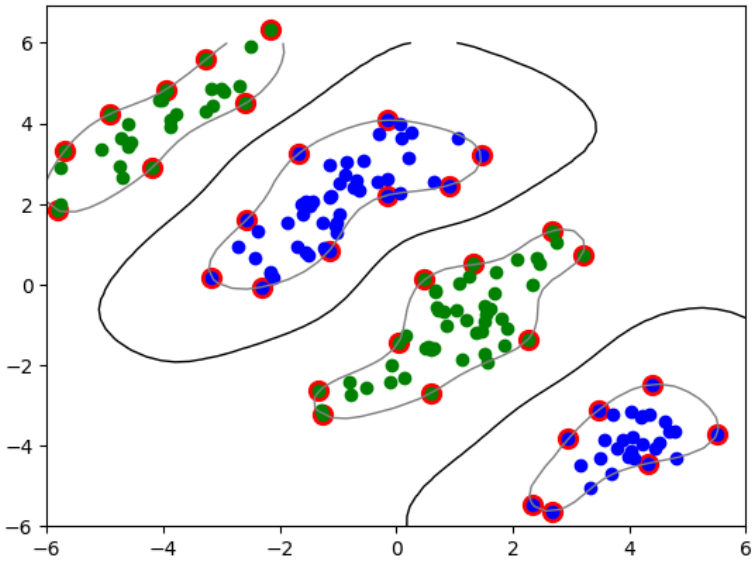
При C=10 и σ = 5:

Количество опорных векторов составит 30 векторов (USV=8, BSV=22), точность классификации останется на уровне 100%, а график будет выглядеть следующим образом.



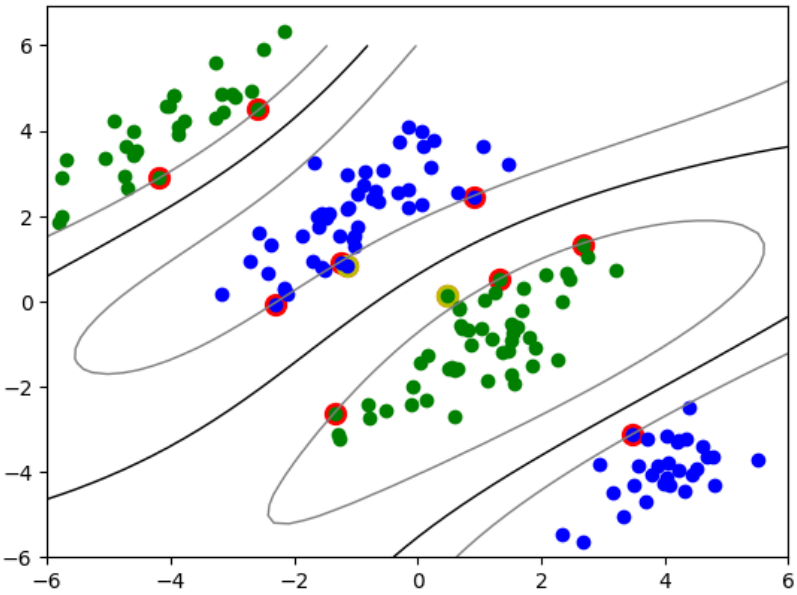
При C=100 и σ = 1:

Количество опорных векторов составит 33 вектора (USV=33, BSV=0), точность классификации останется на уровне 100%, а график будет выглядеть следующим образом.



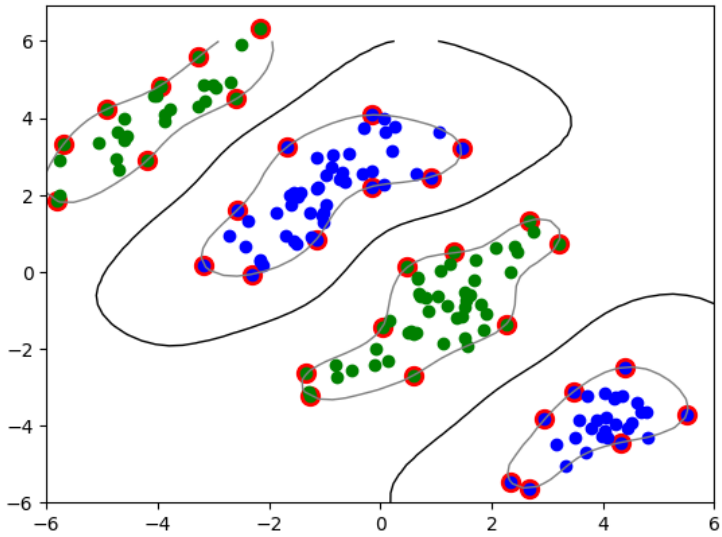
При C=100 и σ = 5:

Количество опорных векторов составит 11 векторов (USV=9, BSV=2), точность классификации останется на уровне 100%, а график будет выглядеть следующим образом.



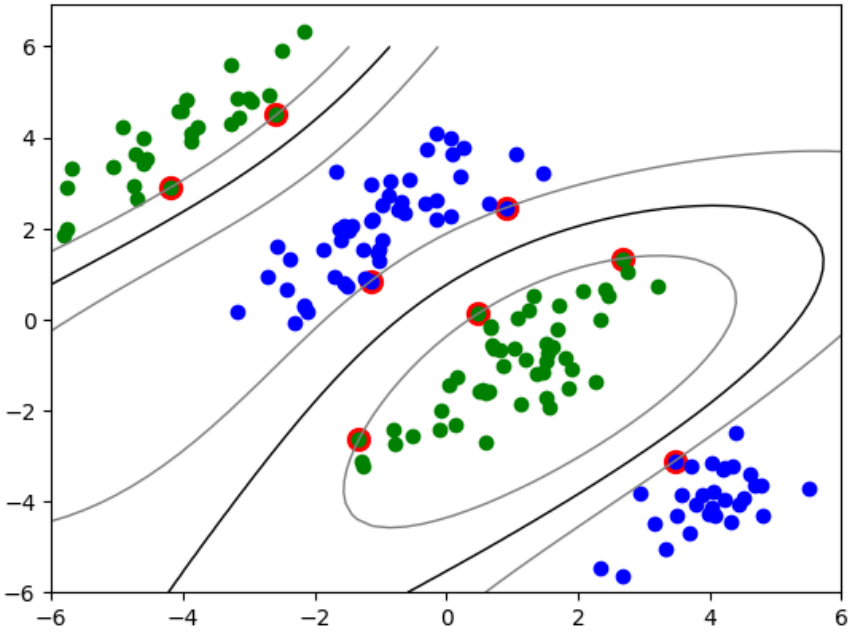
При C=1000 и σ = 1:

Количество опорных векторов составит 33 вектора (USV=33, BSV=0), точность классификации останется на уровне 100%, а график будет выглядеть следующим образом.



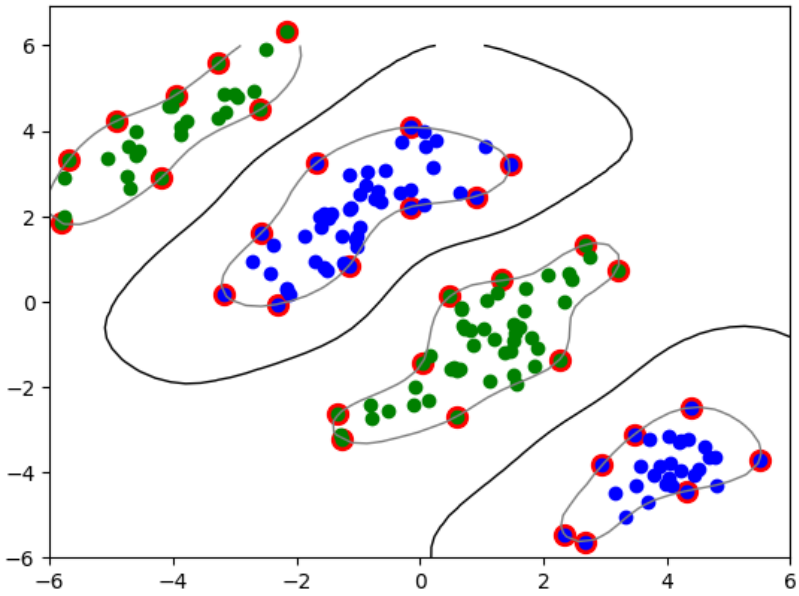
При C=1000 и σ = 5:

Количество опорных векторов составит 8 векторов (USV=8, BSV=0), точность классификации останется на уровне 100%, а график будет выглядеть следующим образом.



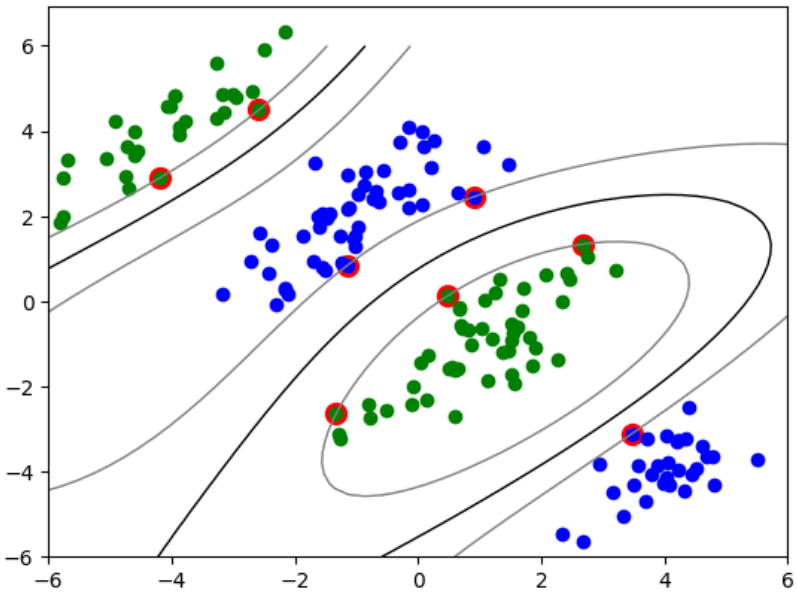
При C=10000 и σ = 1:

Количество опорных векторов составит 33 вектора (USV=33, BSV=0), точность классификации останется на уровне 100%, а график будет выглядеть следующим образом.



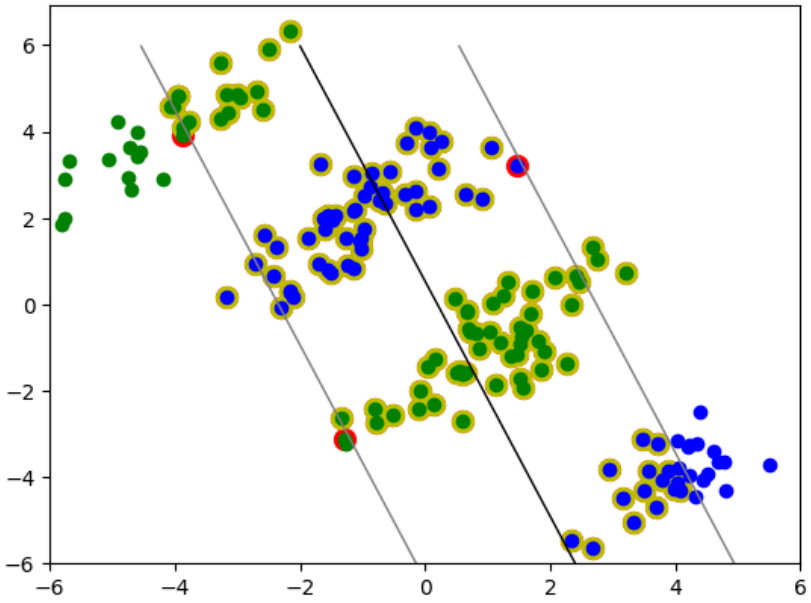
При C=10000 и σ = 5:

Количество опорных векторов составит 8 векторов (USV=8, BSV=0), точность классификации останется на уровне 100%, а график будет выглядеть следующим образом.

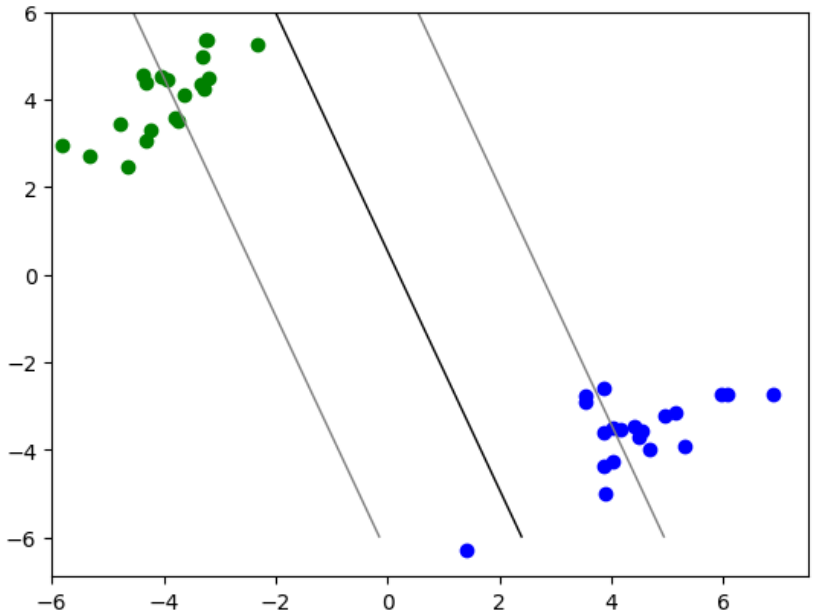


Также попробуем разделить данные через линейное ядро с изменением параметров C.

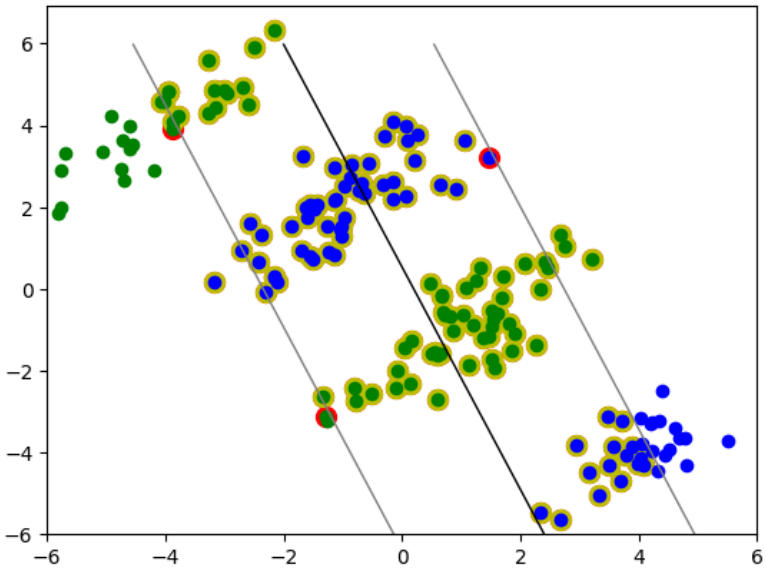
При C = 1: количество опорных векторов 131 (USV=3, BSV=128).



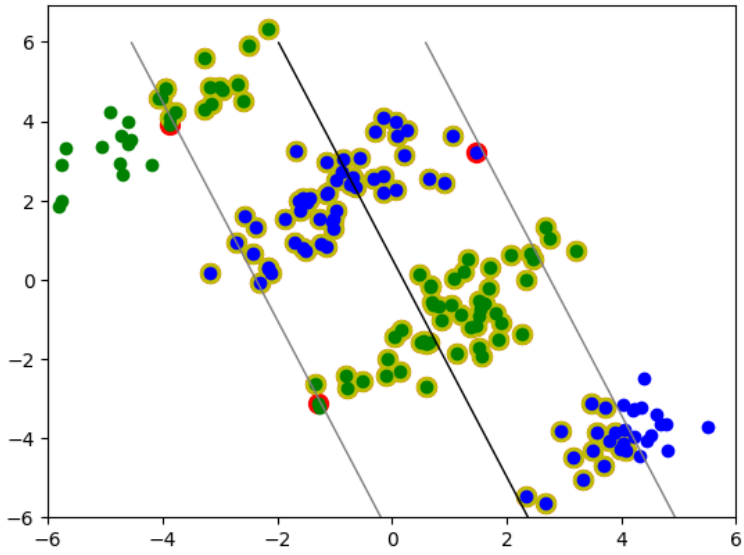
Точность классификации составила 97,5%, но этому параметру не стоит доверять, так как тестовая выборка взята была как хорошо разделенные данные (первые 10% и последние 10% данных).



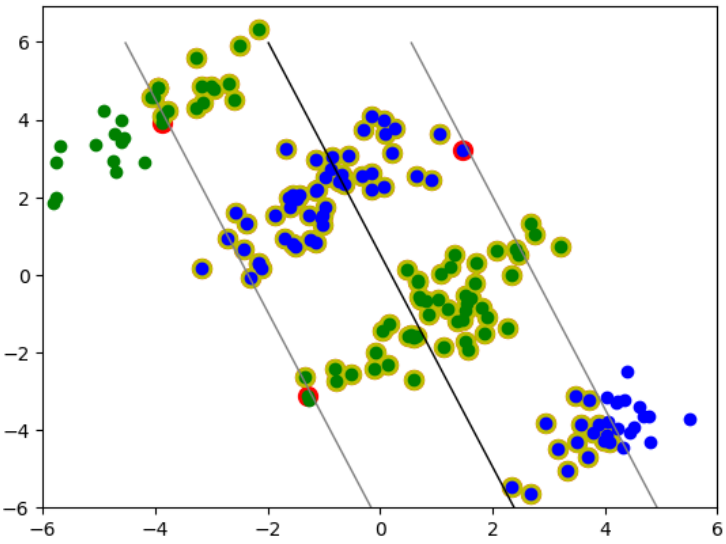
При C = 0.1: количество опорных векторов – 131 (USV=3, BSV=128).



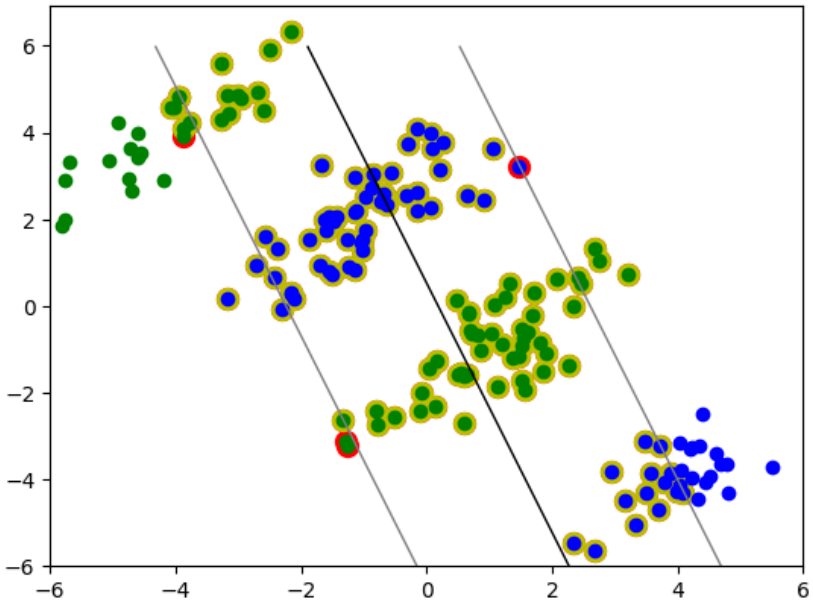
При C = 10: количество опорных векторов – 131 (USV=3, BSV=128).



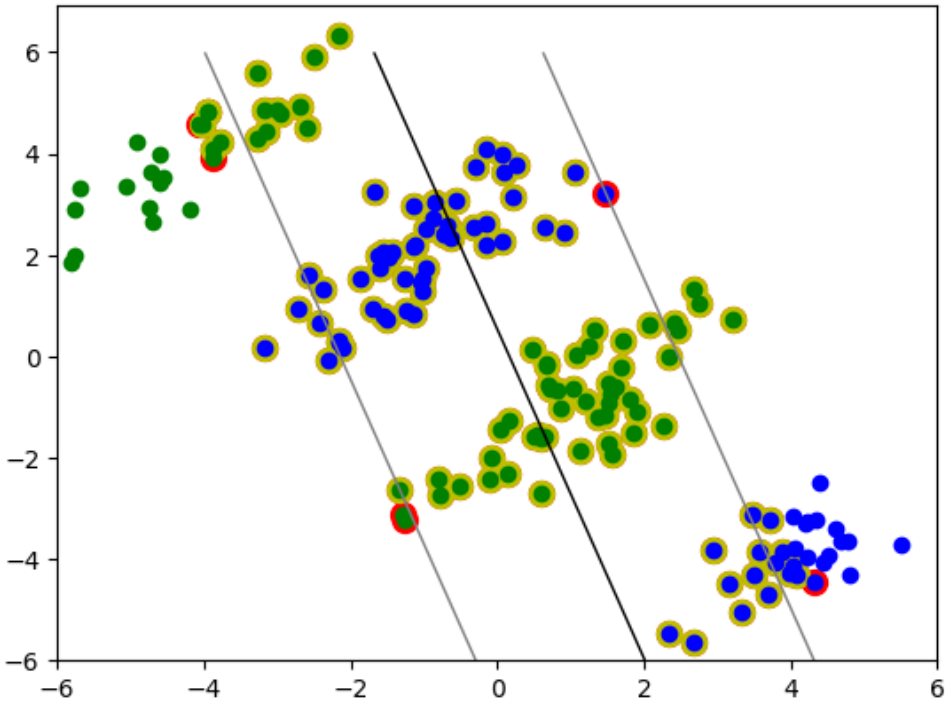
При C = 100: количество опорных векторов – 131 (USV=3, BSV=128).



При C = 1000: количество опорных векторов – 132 (USV=4, BSV=128).

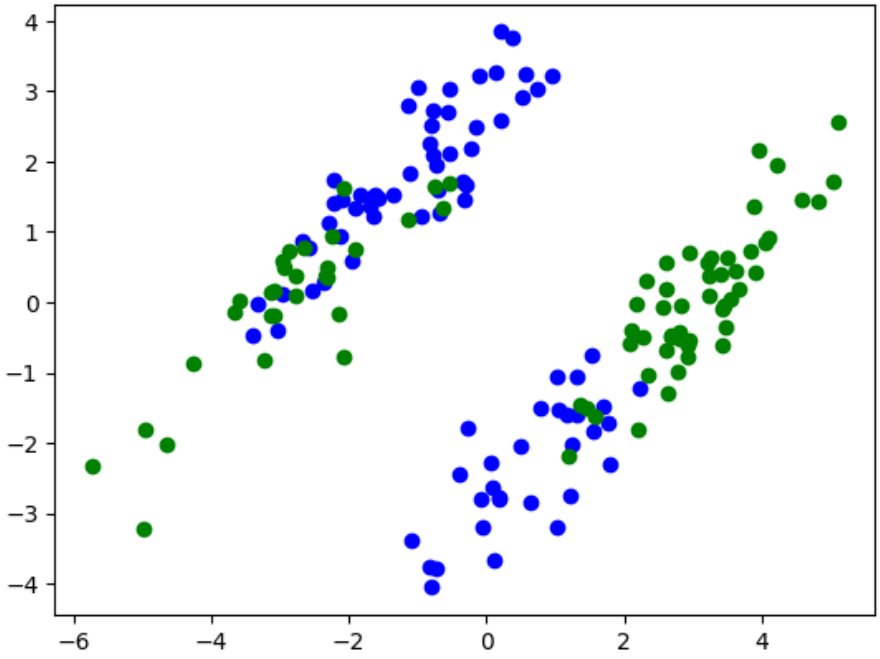


При C = 10000: количество опорных векторов – 133 (USV=6, BSV=127).



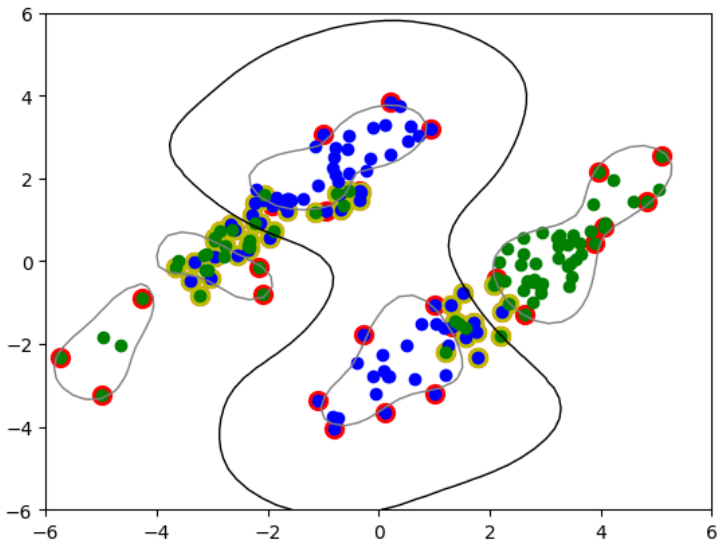
Внесем изменения в сгенерированные данные так, чтобы было пересечение между группами.

```
#Генерация данных
mean1 = [-1, 2]
mean2 = [3, 0]
mean3 = [1, -2]
mean4 = [-3, 0]
cov = [[1.0,0.8], [0.8, 1.0]]
X1 = np.random.multivariate_normal(mean1, cov, 50)
X1 = np.vstack((X1, np.random.multivariate_normal(mean3, cov, 50)))
y1 = np.ones(len(X1))
X2 = np.random.multivariate_normal(mean2, cov, 50)
X2 = np.vstack((X2, np.random.multivariate_normal(mean4, cov, 50)))
y2 = np.ones(len(X2)) * -1
```



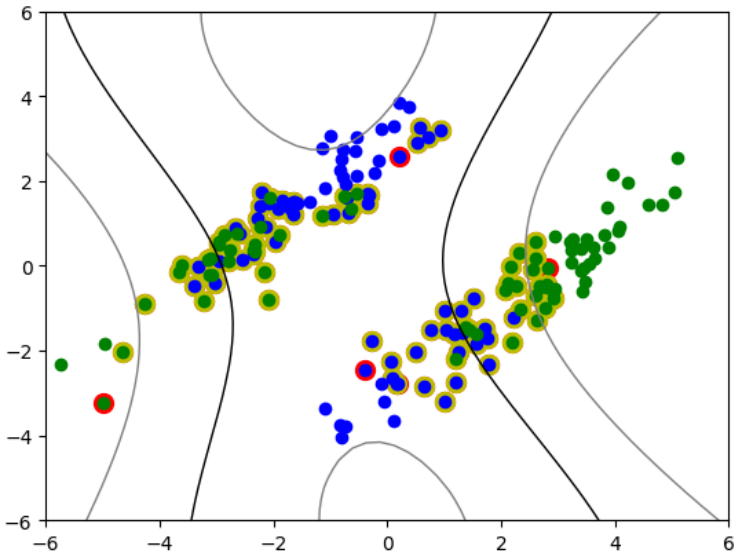
Для начала построим SV-классификатор с гауссовым ядром с **параметрами C=1 и σ=1.**

Количество опорных векторов составило 73 векторов (USV=25, BSV=48). Точность классификации – 92,5%.



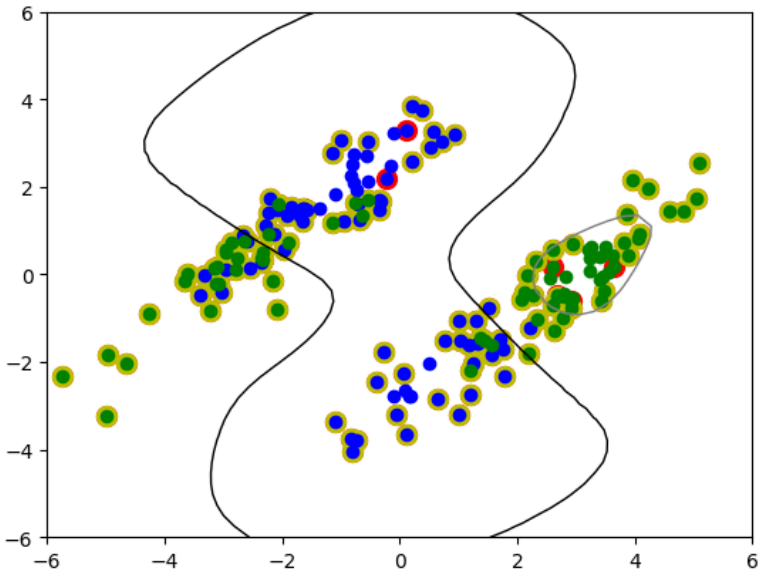
При C=1 и σ=5:

Количество опорных векторов составило 103 вектора (USV=5, BSV=98). Точность классификации – 57,5%.



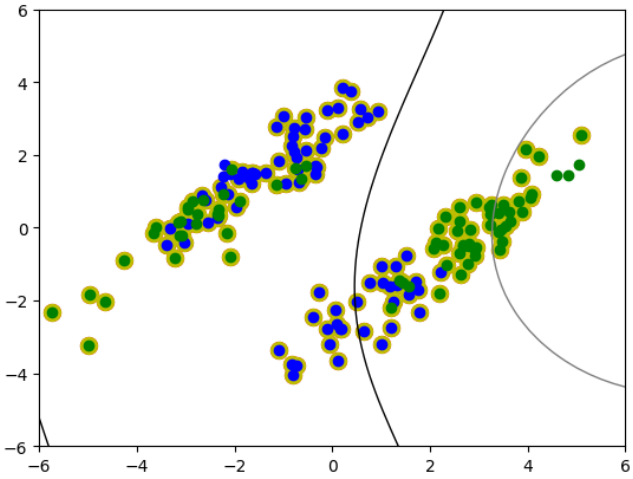
При C=0.1 и σ=1:

Количество опорных векторов составило 129 векторов (USV=7, BSV=122). Точность классификации – 92,5%.



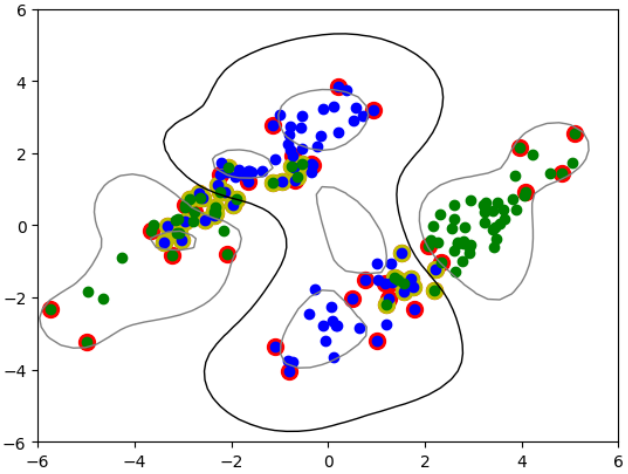
При C=0.1 и σ=5:

Количество опорных векторов составило 154 вектора (USV=0, BSV=154). Точность классификации – 12,5%.



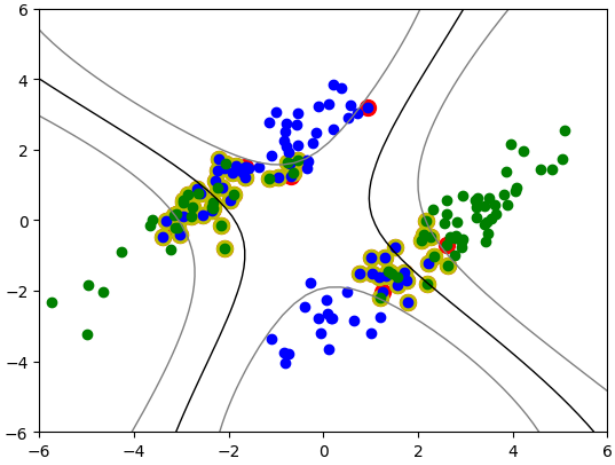
При C=10 и σ=1:

Количество опорных векторов составило 68 векторов (USV=32, BSV=36). Точность классификации – 92,5%.



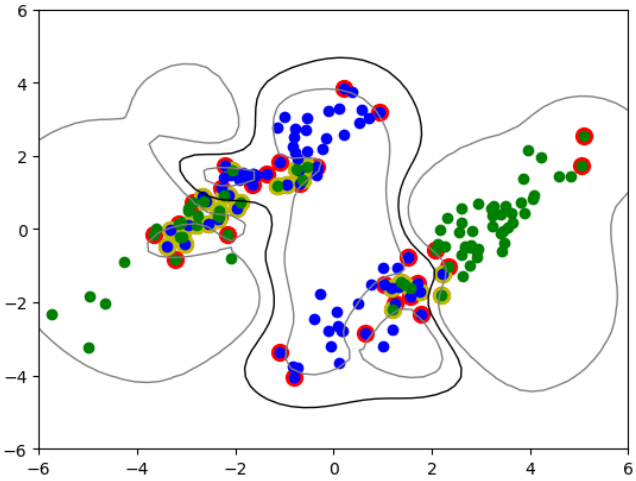
При C=10 и σ=5:

Количество опорных векторов составило 69 векторов (USV=6, BSV=63). Точность классификации – 90%.



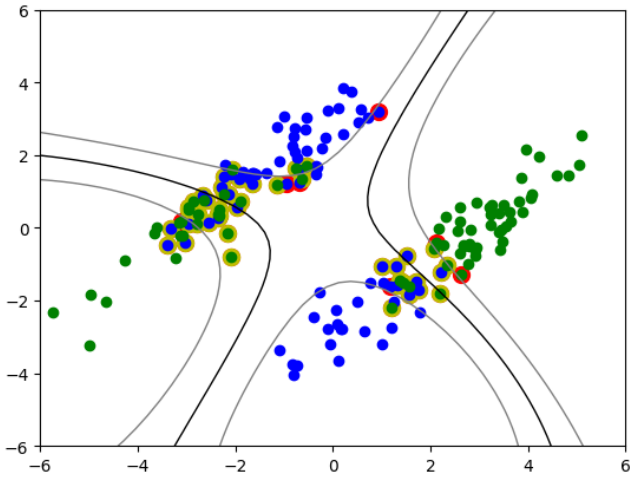
При C=100 и σ=1:

Количество опорных векторов составило 61 вектор (USV=28, BSV=33). Точность классификации – 90%.



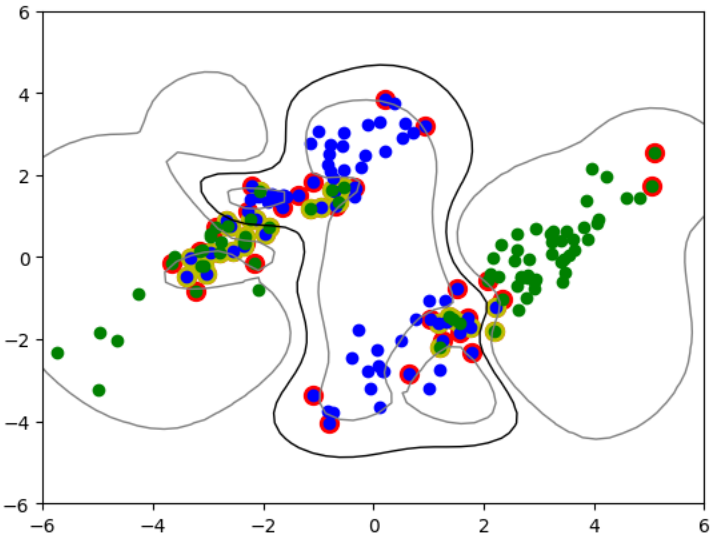
При C=100 и σ=5:

Количество опорных векторов составило 56 векторов (USV=9, BSV=47). Точность классификации – 70%.



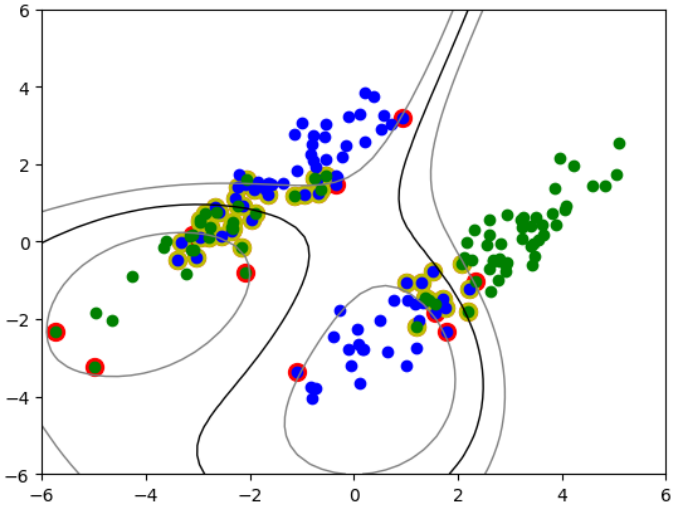
При C=1000 и σ=1:

Количество опорных векторов составило 62 вектора (USV=35, BSV=27). Точность классификации – 82,5%.



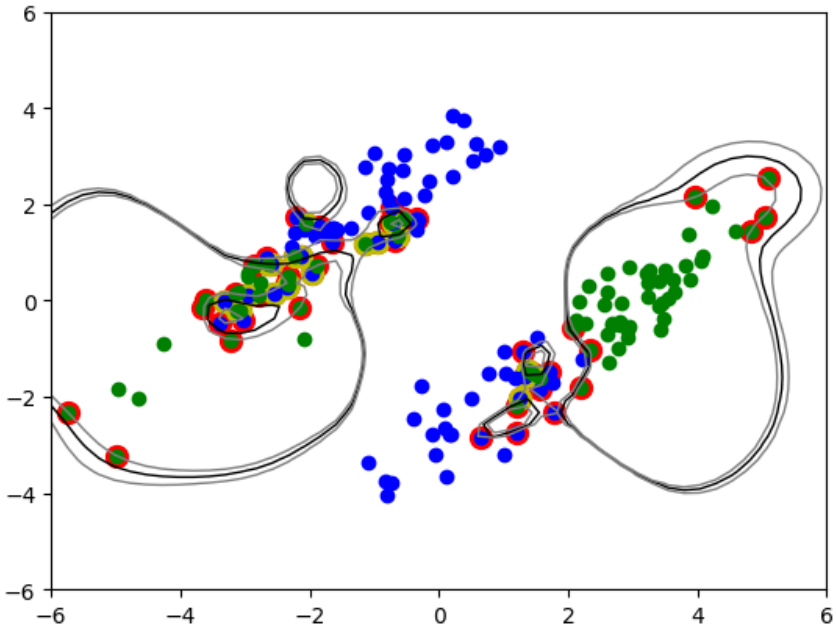
При C=1000 и σ=5:

Количество опорных векторов составило 57 векторов (USV=11, BSV=46). Точность классификации – 90%.



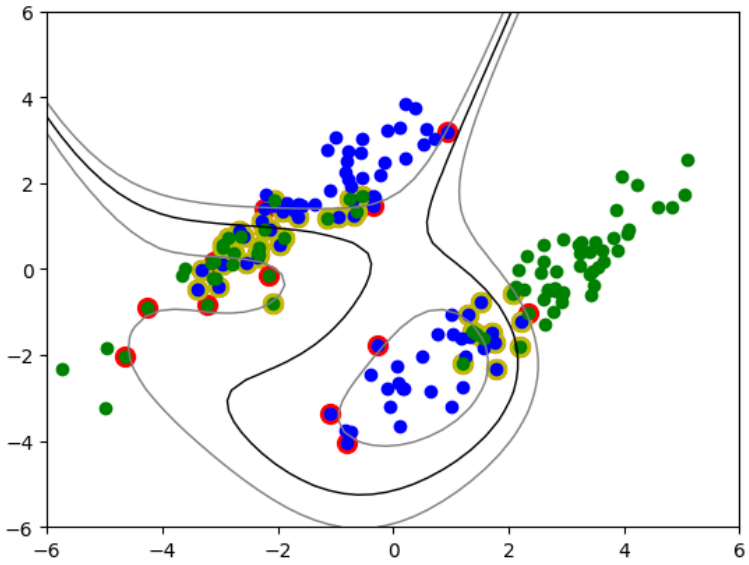
При C=10000 и σ=1:

Количество опорных векторов составило 59 векторов (USV=38, BSV=21). Точность классификации – 80%.



При C=10000 и σ=5:

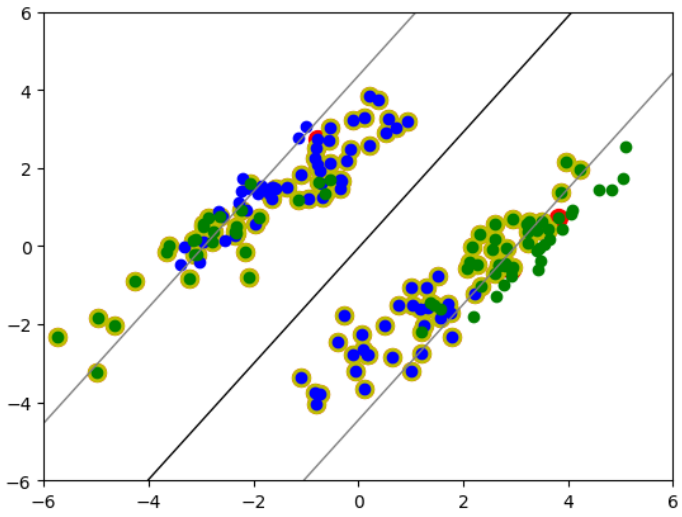
Количество опорных векторов составило 56 векторов (USV=12, BSV=44). Точность классификации – 90%.



Попробуем разделить данные с помощью линейного ядра, изменяя штрафной параметр C.

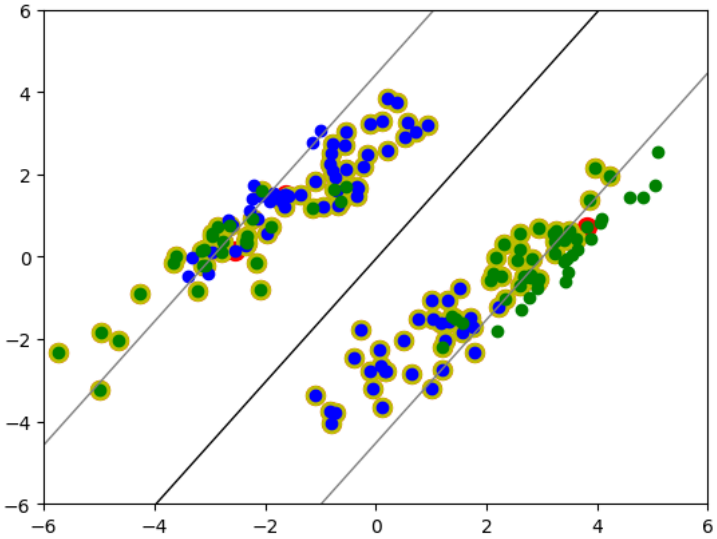
При C=1:

Количество опорных векторов составило 125 векторов (USV=3, BSV=124). Точность классификации – 0%.



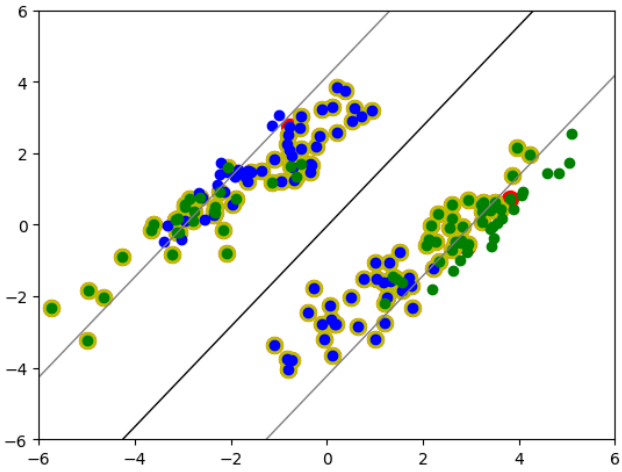
При C=0.1:

Количество опорных векторов составило 127 векторов (USV=3, BSV=124). Точность классификации – 0%.



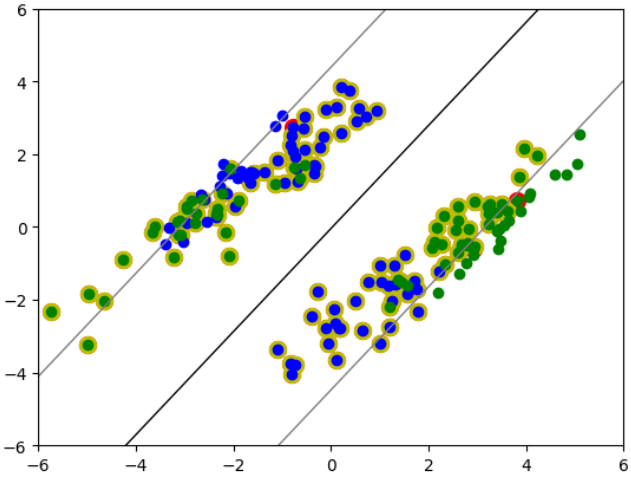
При C=10:

Количество опорных векторов составило 125 векторов (USV=3, BSV=122). Точность классификации – 0%.



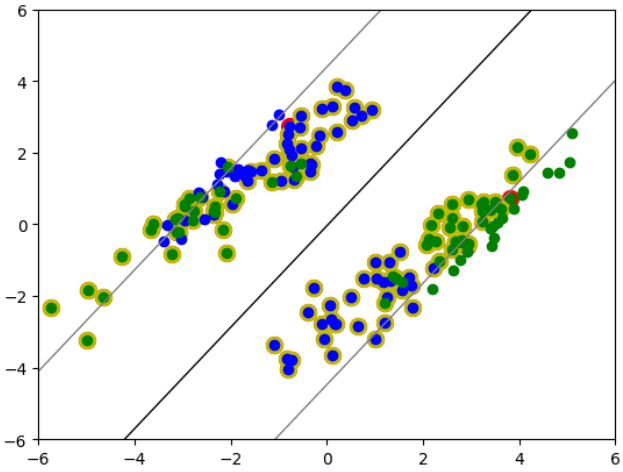
При C=100:

Количество опорных векторов составило 125 векторов (USV=4, BSV=121). Точность классификации – 0%.



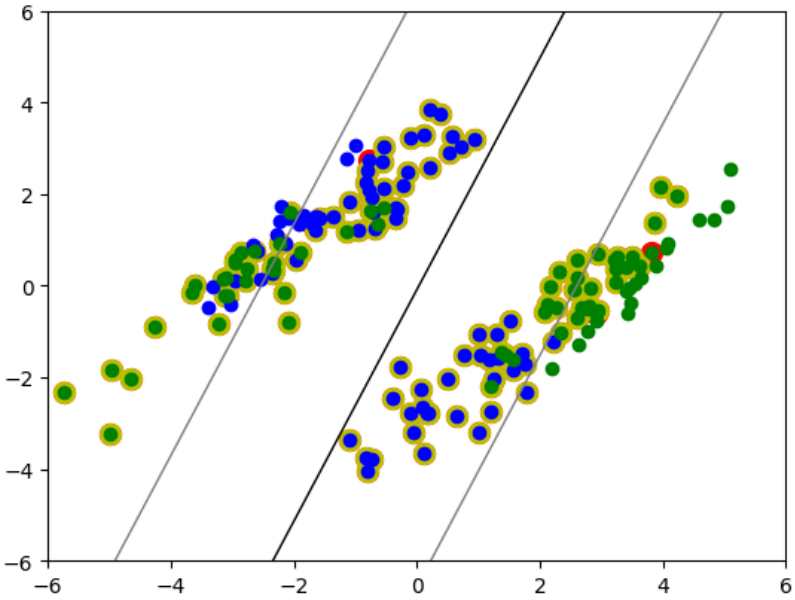
При C=1000:

Количество опорных векторов составило 125 векторов (USV=3, BSV=122). Точность классификации – 0%.



При C=10000:

Количество опорных векторов составило 125 векторов (USV=4, BSV=121). Точность классификации – 0%.



Далее по полученным результатам сделаем выводы.

4. Выводы

По результатам проведенного исследования можно сделать следующие выводы.

1. Для построения правильной SV-машины начальные данные (обучающая выборка) должны быть хорошо разделенными в целях дальнейшего прогноза входных значений. Исследование показало, что при этом точность классификации достигается 100%.

2. Линейная классификация подходит только для данных, которые предварительно хорошо линейно разделены.

3. Если данные плохо разделены, но при этом допускается их линейное разделение, то в этом случае необходимо ввести штрафной параметр C. Исследование показало, что чем выше значение параметра C, тем меньшее количество опорных векторов включается в классификатор и тем меньше ширина допустимый полосы (M). При этом точность классификации оставалась на уровне 95%. Также, чем дальше мы увеличиваем значения параметра C, тем меньше результат отличается от прошлого (например, при C=1000 и C=10000 мы получили идентичные значения наблюдаемых параметров).

4. Для данных, которые разделены нелинейно, необходимо использовать ядерные функции, которые позволяют построить нелинейный SV-классификатор. Если разделить данные линейно, то точность классификации тестовой выборки будет составлять 85-95%, а параметр C отвечал не только за ширину полосы, но также за наклон. Гауссово ядро позволило достичь 100% уровня точности, а также уменьшить количество опорных векторов, включенных в классификатор. При этом вводится новый параметр σ, который в свою очередь отвечает за форму и изгиб основной линии разделения. Чем параметр σ больше, тем линия, разделяющая группы, имеет меньший изгиб, а количество опорных векторов больше. То есть меньшее σ позволяет лучше отделить группы друг от друга с помощью «замкнутых кругов».

5. Данный вывод показали результаты исследования плохо нелийнейно разделенных данных. Точность классификации при линейном разделении показывает 0%. При этом с помощью гауссового ядра удалось увеличить точность до 80%. Чем больше был штрафной параметр C, тем меньшее количество опорных векторов было включено в классификатор и тем выше была точность. При меньшем σ=1 точность была выше, чем σ=5, за счет большей изломанности линии функции.

6. В случае с плохо линейно разделенными данными наилучшей моделью можно считать классификатор с параметрами: C=10, M = 0,46, так как достигается оптимальное количество опорных векторов 9 (USV=4, BSV=5).

7. В случае с нелинейно разделенными данными наиболее оптимальной моделью можно считать модель с параметрами: C=1000 и σ = 5, так как достигается оптимальное количество векторов 8 (USV=8, BSV=0).

8. В случае с нелинейно плохо разделенными данными наиболее оптимальной моделью можно считать модель с параметрами: C=100 и σ=1, так как достигается оптимальное количество векторов 61 (USV=28, BSV=33), точность классификации 90%.