Support Vector Clustering (SVC) — Technical Note

Кластеризация объектов произвольной формы; проверка гиперпараметров q, p

Автор работы: А.В.Мальцева

Дата: 21.05.2025

GitHub: mltsvnn

СОДЕРЖАНИЕ

ЦЕЛЬ И КОНТЕКСТ

Цель проекта — продемонстрировать работу алгоритма Support Vector Clustering (SVC, RBF-ядро) для выделения кластеров произвольной формы и оценить влияние гиперпараметров q и p на результат кластеризации.

Практический контекст — задачи сегментации объектов (например, точек продаж или ассортиментных групп) для последующей дифференциации бизнес-правил: ассортиментная матрица по кластерам, квоты отгрузки, параметры пополнения (WOS, safety stock, пороги заказа).

Почему именно SVC: классические методы вроде k-means предполагают «шарообразные» кластеры и часто плохо разделяют сложные формы («луны», «кольца»). SVC строит границу в признаковом пространстве и позволяет выделять кластеры произвольной геометрии, что важно при нелинейных зависимостях.

Данные в проекте синтетические — цель состоит в демонстрации методики и её устойчивости: сравнение числа кластеров, количества SV/BSV, поведения модели при разных q, p.

1. МЕТОДОЛОГИЯ

1.1. Определение границ кластеров

Задача кластеризации заключается в разбиении всей анализируемой совокупности объектов на несколько однородных кластеров.

Суть алгоритма SV-кластеризации (support vector clustering, SVC) состоит в следующем: исходные векторы через гауссово ядро $k(x_i,\ x_j) = \exp(-q||x_i - x_j||^2)$, q>0, отображаются в пространство большей размерности, где ищется минимальная гиперсфера, охватывающая все точки. При обратном отображении на исходное пространство эта сфера распадается на несколько замкнутых контуров — границ кластеров. Для последующей маркировки (cluster labeling) строят матрицу смежности, а кластеры определяют как связанные компоненты индуцированного графа.

Гиперпараметры SVC:

- q — параметр ядра, который управляет масштабом исследуемых данных;
- C —штрафной параметр, ограничивающий долю точек, лежащих вне сферы, контролирующий перекрытие кластеров и выбросы.

Структуру данных анализируют, варьируя q и C, при этом стараясь сохранять минимальное число опорных векторов для получения гладких границ кластеров.

Перейдем к определению границ кластеров.
Пусть $X \subseteq R^p$ – исходное пространство объектов , $\{x_i\} \in X$, i =1…,n, - множество точек этого пространства. Используя нелинейное преобразование Ф из исходного пространства X в некоторое гилбертово пространство признаков большей размерности, будем искать гиперсферу с центром в точке a наименьшего радиуса R, охватывающую в пространстве признаков все исходные точки

$$\varphi:\ \boldsymbol{x} \in \mathcal{X} \ \rightarrow \varphi\ (\boldsymbol{x}) \in H$$

$$\forall\ \boldsymbol{x}\ ,\ \boldsymbol{y} \in \mathcal{X} \ \rightarrow k(\boldsymbol{x}, \boldsymbol{y}) = exp(-q||\boldsymbol{x}-\boldsymbol{y}\ ||^2),\ \ q > 0\ \ (\ q=\frac{1}{2\sigma^2}\ )$$

Постановка исходной задачи**:**

$$\boldsymbol{R^2 \rightarrow \min_a}$$

$$\boldsymbol{||\varphi\ (x_i) - a||^2 \leq R^2,\ \forall\ i \in \{1,\ldots,n\}}$$

$||\cdot||$ – евклидова норма, a – центр сферы

Введем неотрицательные переменные ξi, ослабляющие жесткие ограничения. Точки x_i , которым соответствуют ξi > 0, находятся в пространстве признаков на большем, чем радиус R, расстоянии от центра a, т. е. лежат вне гиперсферы

$$\xi_i \geq 0 \quad \forall i \in \{1, \dots, n\}, C > 0$$

Добавим в целевую функцию сумму ξi > 0, контролируемую штрафным параметром C > 0, и представим окончательную формулировку оптимизационной задачи:

$$\min_{R^2,\ \xi_i,\ \boldsymbol{a}} \left\{ R^2 + C \sum_{i=1}^{n} \xi_i \right\}$$

$$\left|\left|\varphi(\boldsymbol{x}_i) - \boldsymbol{a}\right|\right|^2 \leq R^2 + \xi_i \ , \ \xi_i \geq 0 \quad \forall i \in \{1, \dots, n\}$$

Составим лагранжиан Lp, соответствующий этой задаче:

$$L_p\,(R^2, \xi,\ a,\ \mu,\ \beta) = R^2 + C\,\Sigma_i \xi_i - \Sigma_i\, \mu_i (R^2 + \xi_i - ||\varphi\,(\boldsymbol{x}_i) - \boldsymbol{a}||^2) - \Sigma_i\, \beta_i \xi_i$$

$$\mu_i \geq 0,\ \beta_i \geq 0, \quad \forall\, i \in \{1, \dots, n\}$$

Приведем формулировку прямой задачи в терминах лагранжиана: найти минимум функции Lp по переменным R2, ξ и a при условии, что градиент Lp по всем βi, μiравен нулю. Получаем задачу выпуклого квадратичного программирования, вместо нее можно решить эквивалентную ей двойственную задачу: найти максимум функции Lpпо переменнымμиβпри условии, что градиент лагранжиана по переменным R2, ξ и a равен нулю, βi ≥ 0, μi ≥ 0.Вычислив градиент лагранжиана Lpпо переменным R2, ξ, аи приравняв его нулю, получим

$$\frac{\partial L_p}{\partial R^2} = 0 \rightarrow 1 - \sum_{i=1}^{n} \mu_i = 0 \rightarrow \sum_{i=1}^{n} \mu_i = 1$$

$$\frac{\partial L_p}{\partial \xi_i} = 0 \rightarrow\ C - \mu_i - \beta_i = 0\ \rightarrow C = \mu_i + \beta_i$$

$$\frac{\partial L_p}{\partial \boldsymbol{a}} = 0\ \rightarrow -2 \sum_{i=1}^{n} \mu_i\, \varphi(\boldsymbol{x}_i) + 2\boldsymbol{a} = 0 \rightarrow \boldsymbol{a} = \sum_{i=1}^{n} \mu_i\, \varphi(\boldsymbol{x}_i)$$

Подставив ограничения в лагранжиан, исключим переменные R, ξ и a, скалярные произведения (xi, xj) заменим ядерной функцией k(xi, xj) = (Φ(xi), Φ(xj)) и получим двойственную задачу:

$$\max_{\boldsymbol{\mu}} \boldsymbol{W}:$$

$$\boldsymbol{W}(\mu) = \sum_{i=1}^{n} \boldsymbol{\mu_i k}(\boldsymbol{x_i}, \boldsymbol{x_i}) - \sum_{i=1}^{n} \sum_{j=1}^{n} \boldsymbol{\mu_i \mu_j k}\left(\boldsymbol{x_i}, \boldsymbol{x_j}\right)$$

$$\sum_{i=1}^{n} \boldsymbol{\mu_i} = \mathbf{1}$$

$$\mathbf{0} \leq \boldsymbol{\mu_i} \leq \boldsymbol{C}, \boldsymbol{i} = \mathbf{1}, \ldots, \boldsymbol{n}$$

Таким образом, решение исходной задачи эквивалентно решению задачи квадратичного программирования, удовлетворяющего условиям Каруша–Куна–Таккера.

Условия комплементарности:

$$\beta_i \, \xi_i = 0$$

$$\mu_i \, (R^2 + \xi_i - ||\varphi \, (\boldsymbol{x}_i) - \boldsymbol{a}||^2) = 0$$

Тогда:

1) Bounded Support Vectors (BSVs) – находится вне гиперсферы
При $\mu_i = C$ и $\xi_i > 0$

2) Support Vectors (SVs) – находятся на поверхности гиперсферы
При $\xi_i = 0$, $0 < \mu_i < C$

3) Other Vectors (OVs) – внутри гиперсферы (или на поверхности)
При $\xi_i = 0$, $\mu_i = 0$

Применим RBF-ядро: вся сложность контуров кластеров сводится к регулированию одного параметра q, задающему ширину ядра. Для любой точки $x \in \mathcal{X}$ введём:

$$r^2(x) = \|\Phi(x) - a\|^2,$$

подставив разложение:

$$\boldsymbol{a} = \sum_{i=1}^{n} \mu_i \, \varphi(\boldsymbol{x}_i),$$

получим:

$$\boldsymbol{r^2(x) = k(x, x) - 2\Sigma_i \, \mu_i \, k(x_i, x) + \Sigma_{i,j} \, \mu_i \, \mu_j \, k(x_i, x_j)}$$

Все опорные векторы в пространстве признаков находятся на расстоянии R от центра сферы a.

Отсюда

$$R = r(\boldsymbol{x}_i) \mid \forall \, \boldsymbol{x}_i \in \text{SVs}$$

В исходном пространстве контуры определяются множеством векторов x, удовлетворяющих условию

$$\{\boldsymbol{x} \in \mathcal{X} : r(\boldsymbol{x}) = R\}$$

1.2. Маркировка кластеров (cluster labeling)

После того как границы кластеров определены, переходим к их маркировке. Используем геометрический подход: если две точки принадлежат разным кластерам, то любой путь между ними в пространстве признаков должен выходить за пределы сферы радиуса R с центром в точке a. Отсюда для каждой пары xi, xj, находящихся внутри или на поверхности сферы, вводят элементы матрицы смежности

$$A_{ij} = \begin{cases} 1, & \text{если } r(\mathbf{y}) \leq R \quad \forall\, \boldsymbol{y} \in S \\ 0, & \text{иначе} \end{cases}$$

1.3. Подбор значений параметров q и p

В силу ограничений двойственной задачи верхняя граница доли связанных опорных векторов определяется величиной p = 1/(nC). Подбор параметров рекомендуется начинать со значений:

$$p = \frac{1}{n}, \quad q = \frac{1}{\max_{i,j} \|x_i - x_j\|^2},$$

2. РЕАЛИЗАЦИЯ И ЭСПЕРИМЕНТЫ

2.1. Построение SVC

Для реализации алгоритма SVC был построен класс SV_cluster. В конструкторе этого класса задаются все ключевые гиперпараметры — коэффициент q, управляющий шириной RBF-ядра, и величина p, ограничивающая долю «связных» опорных векторов, из которой автоматически выводится параметр жёсткости $C=1/np$.

Метод _kernel реализует гауссово (RBF) ядро

```
    def _kernel(self, p1, p2):
        return np.exp(-self.q * np.sum((np.asarray(p1) - np.asarray(p2))**2))
```

Далее двойным циклом формируется матрица K

```
        K_gram = np.zeros((n_samples, n_samples))
        for i in range(n_samples):
            for j in range(i, n_samples):
                val = self._kernel(self.X_train_[i], self.X_train_[j])
                K_gram[i, j] = val
                K_gram[j, i] = val
```

Решаем задачу:

```
        epsilon_reg = 1e-9
        P_qp_cvx = matrix(2 * K_gram + epsilon_reg * np.eye(n_samples))
        q_qp_cvx = matrix(-np.ones(n_samples))
```

Находим опорные вектора:

```
        self.support_indices_ = np.where(self.betas_ > self.sv_tol)[0]
        self.support_vectors_ = self.X_train_[self.support_indices_]
        self.n_sv_ = len(self.support_indices_)
        self.n_bsv_ = np.sum(self.betas_[self.support_indices_] >=
                                              self.C_param - self.sv_tol)
```

Считаем параметры сферы:

```
        self.Ca_ = self.betas_.T @ K_gram @ self.betas_

        min_Dx_sq_for_sv = float('inf')
        for sv_idx in self.support_indices_:
            kernels_sv_X =
self._kernel_point_to_matrix(self.X_train_[sv_idx], self.X_train_)
            sum_beta_K_sv = np.sum(self.betas_ * kernels_sv_X)
            Dx_sq = 1.0 - 2 * sum_beta_K_sv + self.Ca_
            if Dx_sq < min_Dx_sq_for_sv:
                min_Dx_sq_for_sv = Dx_sq
        self.R_squared_ = min_Dx_sq_for_sv
```

Делим на кластеры на основе графа связности:

```
        self._build_clusters_graph(self.X_train_)
```

После того как параметры найдены, метод _get_Dx_squared и _is_in_sphere реализуем:

$$r^2(x) = k(x, x) - 2\Sigma_i \mu_i k(x_i, x) + \Sigma_{i,j} \mu_i \mu_j k(x_i, x_j)$$

```
    def _get_Dx_squared(self, x_point):
        kernels_x_Xtrain = self._kernel_point_to_matrix(x_point,
self.X_train_)
        sum_beta_K_x_Xtrain = np.sum(self.betas_ * kernels_x_Xtrain)
        return 1.0 - 2 * sum_beta_K_x_Xtrain + self.Ca_

    def _is_in_sphere(self, x_point, radius_tol=1e-9):
        if self.R_squared_ is None: return False
        return self._get_Dx_squared(x_point) <= self.R_squared_ + radius_tol
```

_build_clusters_graph реализует маркировку кластеров: если любой путь между двумя точками остаётся внутри сферы, они относятся к одному кластеру.

```
    def _build_clusters_graph(self, X):
        n_samples = X.shape[0]
        adj_matrix = np.zeros((n_samples, n_samples), dtype=int)

        point_in_sphere_flags = np.array([self._is_in_sphere(X[i]) for i in
range(n_samples)])

        for i in range(n_samples):
            for j in range(i + 1, n_samples):
                segment_is_good = True
                for k_seg in range(1, self.num_segments_check + 1):
                    t = k_seg / (self.num_segments_check + 1.0)
                    point_on_segment = (1 - t) * X[i] + t * X[j]
                    if not self._is_in_sphere(point_on_segment):
                        segment_is_good = False
                        break

                if segment_is_good:
                    adj_matrix[i, j] = 1
                    adj_matrix[j, i] = 1

        self.n_clusters_, self.labels_ = connected_components(
            csgraph=adj_matrix, directed=False, return_labels=True
        )
```

После того как модель обучена выполняется визуализация. Сначала определяется, сколько уникальных меток получилось, зависимости от этого выбирается карта цветов:

```
        num_unique_labels = len(np.unique(self.labels_))
        cmap_name = 'viridis' # По умолчанию
        if num_unique_labels > 0:
            if num_unique_labels == 1: cmap_name = 'winter'
            elif num_unique_labels <= 10: cmap_name = 'tab10'
            else: cmap_name = 'nipy_spectral' # Если много кластеров

        colors_map = plt.cm.get_cmap(cmap_name, max(1, num_unique_labels))

        # Рисуем точки данных
        ax.scatter(X_plot[:, 0], X_plot[:, 1], c=self.labels_,
cmap=colors_map,
                   s=40, edgecolors='black', alpha=0.85, zorder=2)
```

Производится визуализация SVs и BSVs при наличии

```
        if self.support_vectors_ is not None and len(self.support_vectors_)
> 0:
            ax.scatter(self.support_vectors_[:, 0], self.support_vectors_[:,
1],
                       s=160, facecolors='none', edgecolors='red',
linewidths=1.8,
                       label=f'SVs: {self.n_sv_}', zorder=4, marker='o')
            if self.n_bsv_ > 0:
                bsv_mask = self.betas_[self.support_indices_] >=
self.C_param - self.sv_tol
                actual_bsv_points = self.support_vectors_[bsv_mask]
                if len(actual_bsv_points) > 0:
                    ax.scatter(actual_bsv_points[:, 0], actual_bsv_points[:,
1],
                               s=220, facecolors='none', edgecolors='blue',
linewidths=1.8, linestyle=':',
                               label=f' BSVs: {self.n_bsv_}', zorder=5,
marker='X')
```

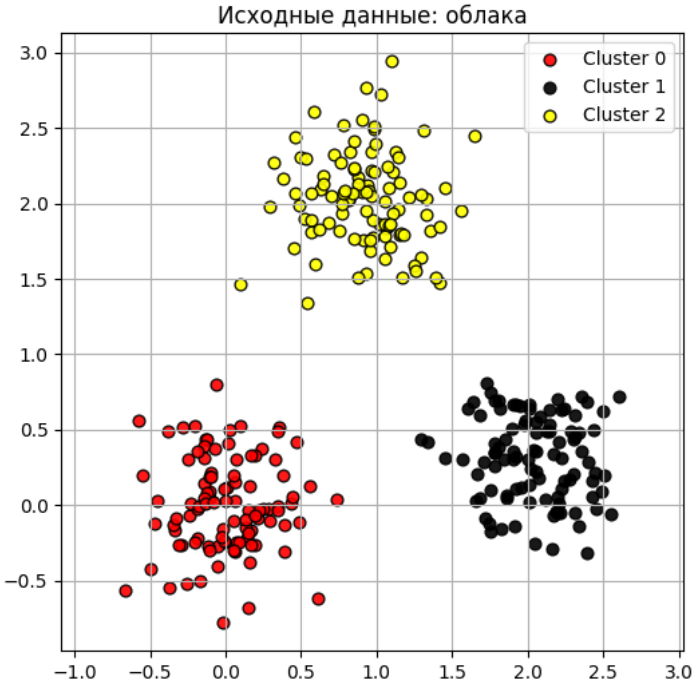
Если радиус сферы рассчитан, строится сетка и методом ax.contour проводится линия уровня, охватывающая все наблюдения внутри объединённого кластера.

```
        if self.R_squared_ is not None:
            x_min, x_max = X_plot[:, 0].min() - 0.5, X_plot[:, 0].max() +
0.5
            y_min, y_max = X_plot[:, 1].min() - 0.5, X_plot[:, 1].max() +
0.5
            xx, yy = np.meshgrid(np.linspace(x_min, x_max, 70),
                                 np.linspace(y_min, y_max, 70))
```

```
            grid_points = np.c_[xx.ravel(), yy.ravel()]
            Z_values_Dx_sq = np.array([self._get_Dx_squared(p) for p in
grid_points])
            Z_contour = (Z_values_Dx_sq - self.R_squared_).reshape(xx.shape)
            ax.contour(xx, yy, Z_contour, levels=[0], linewidths=2,
colors='black', zorder=3)
```

2.2. Подготовка модельных данных: облака

Для начала были подготовлены данные, состоящие из трёх хорошо разделимых кластеров из нормальных распределений по 100 объектов в каждом.



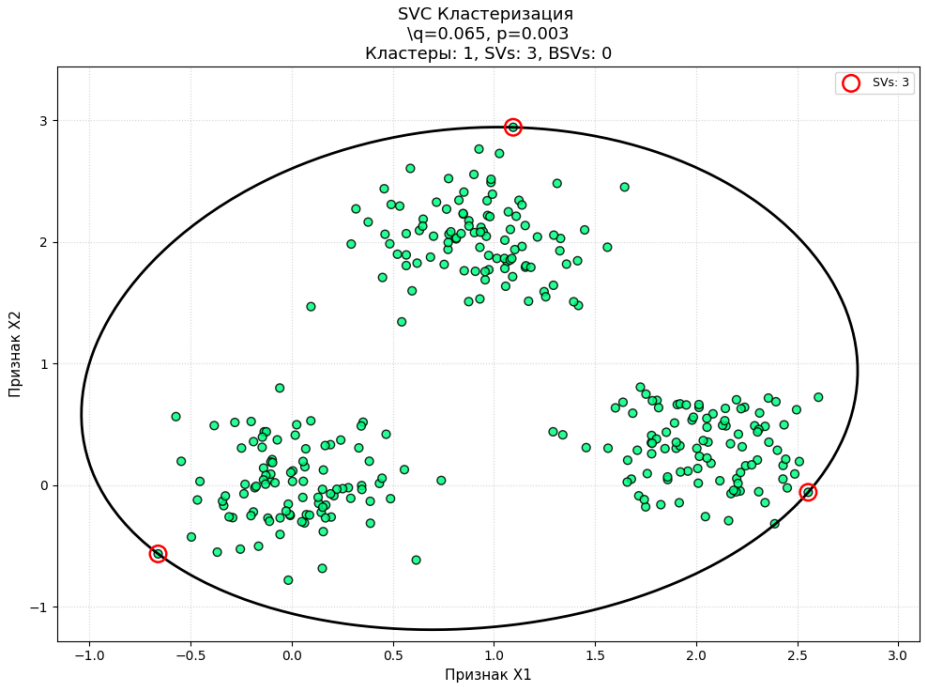
2.3. Облака: подбор значений гиперпараметров SVC-алгоритма

Подбор параметров рекомендовано начинать со значений:

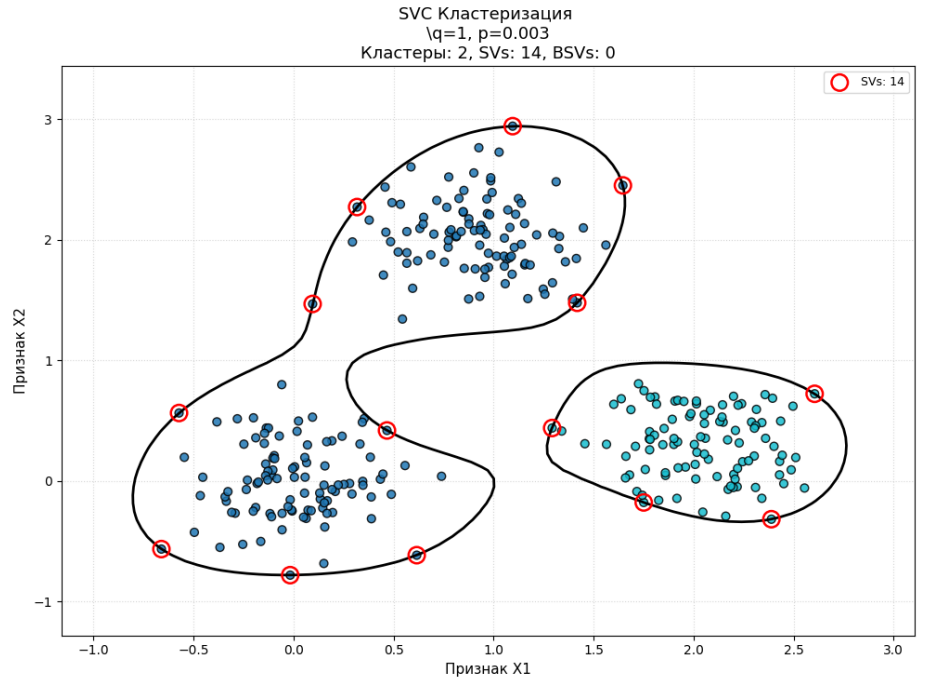
$$p = \frac{1}{n}, \quad q = \frac{1}{\max_{i,j} \|x_i - x_j\|^2},$$

Соответственно:
- стартовое значение p=0.003
- стартовое значение q= 0.065

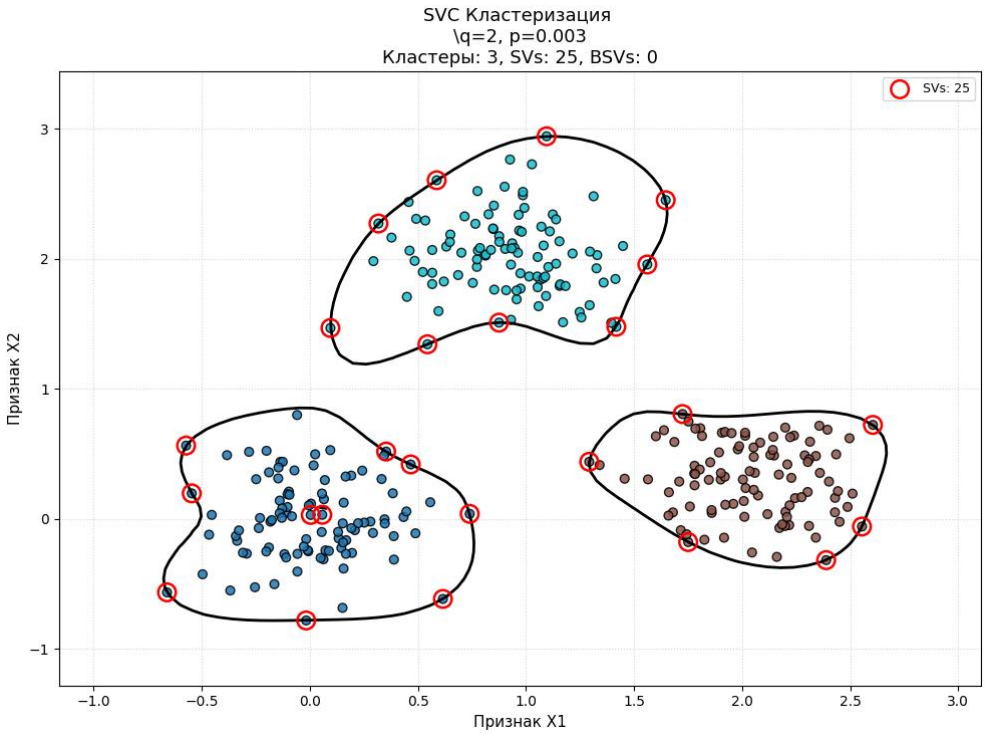
В результате действительно наблюдается небольшое количество опорных векторов: всего три SV (BSV отсутствуют) и один большой кластер, охватывающий все объекты:



При увеличении q до 1 видим отделение второго кластера, количество опорных векторов растет до 14:



При q=2 число SVs дорастает до 25 и наблюдаются три четко разделимых кластера.



Таким образом, по мере увеличения q при малом p наблюдается последовательный рост числа выделяемых кластеров и опорных векторов, при этом граница выглядит достаточно сглаженной, BSV отсутсвуют.

Запустим решетку по p и q. Диапазоны значений:

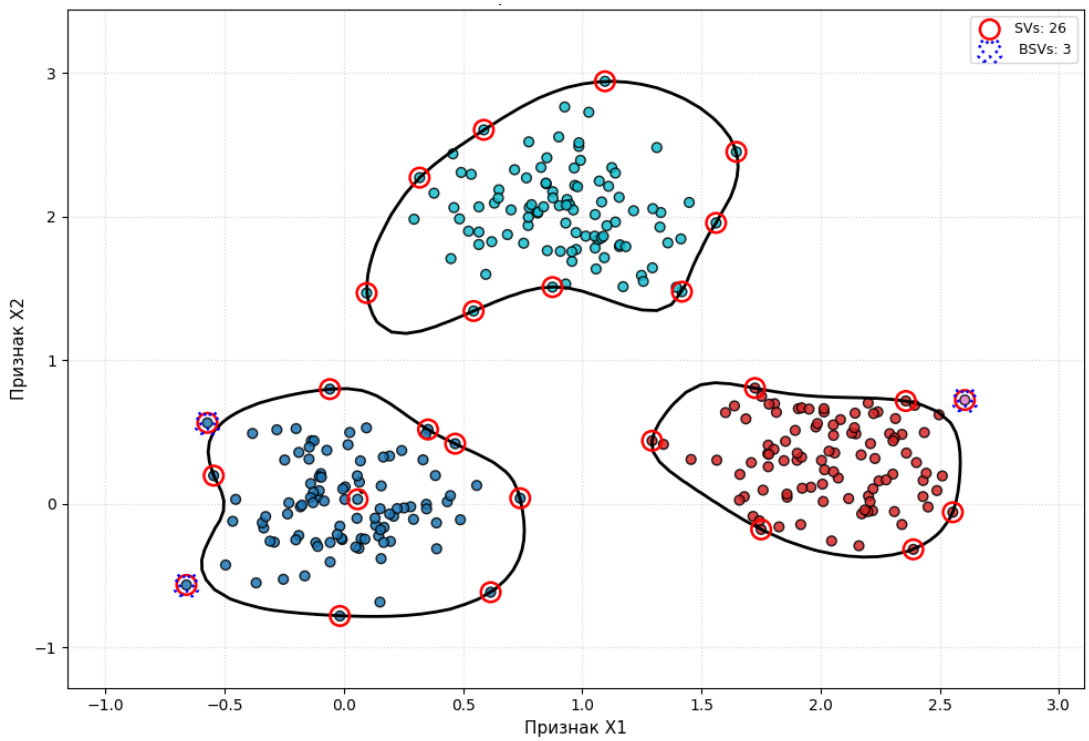
```
for p in [0.003, 0.05, 0.1, 0.2]:
  for q in [0.065, 1, 2, 3, 4, 5]:
```

Результат представим в виде таблицы:

q	p	Количество кластеров	SVs	BSVs
0.065	0.003	1	3	0
1.000	0.003	2	14	0
2.000	0.003	3	25	0
3.000	0.003	3	31	0
4.000	0.003	3	35	0
5.000	0.003	3	43	0
0.065	0.050	3	15	12
1.000	0.050	8	22	9
2.000	0.050	3	26	3
3.000	0.050	3	31	0
4.000	0.050	3	36	0
5.000	0.050	3	44	0

0.065	0.100	6	35	32
1.000	0.100	16	38	27
2.000	0.100	12	41	27
3.000	0.100	12	46	21
4.000	0.100	12	50	22
5.000	0.100	12	55	18
0.065	0.200	11	52	49
1.000	0.200	34	54	45
2.000	0.200	24	59	42
3.000	0.200	24	61	41
4.000	0.200	21	66	39
5.000	0.200	20	67	37

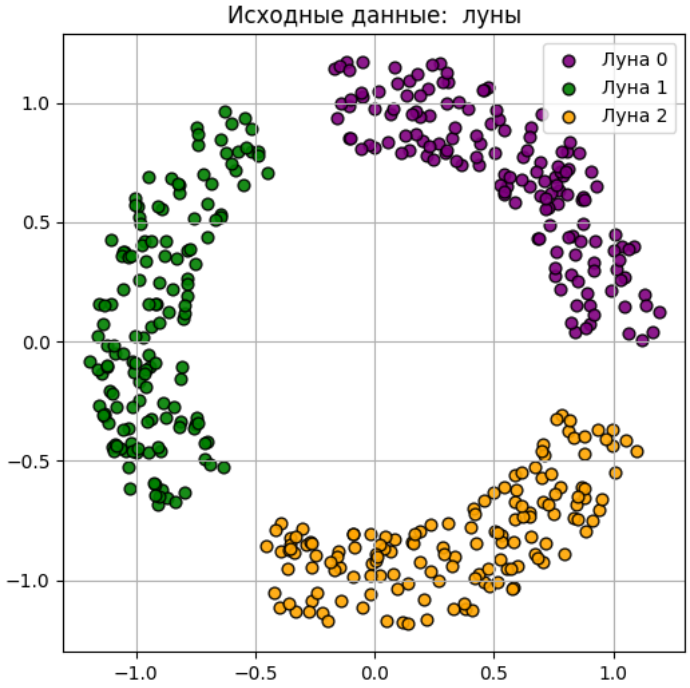
Видим, что при увеличении p уже до 0.05 мы сразу получаем три кластера, но уже с заметным числом BSV (12). При q=1 наблюдается излишнее усложнение: количество кластеров увеличивается до 8, хотя мы знаем, что в данных три естественных группы. При q=2 алгоритм снова различает три кластера, SVs при этом 26, SVs 3:



При дальнейшем увеличении q SVs и BSVs выровнялись по аналогии с p=0.003. В случае с p=0.1, даже при самом небольшом q алгоритм дробит данные на целых 6 кластеров, при q=1 число кластеров взлетает до 16, SV и BSV до 38 и 27 соответственно. Дальнейшее увеличение p разбивает каждое локальное скопление точек на свой кластер, что является переобучением.

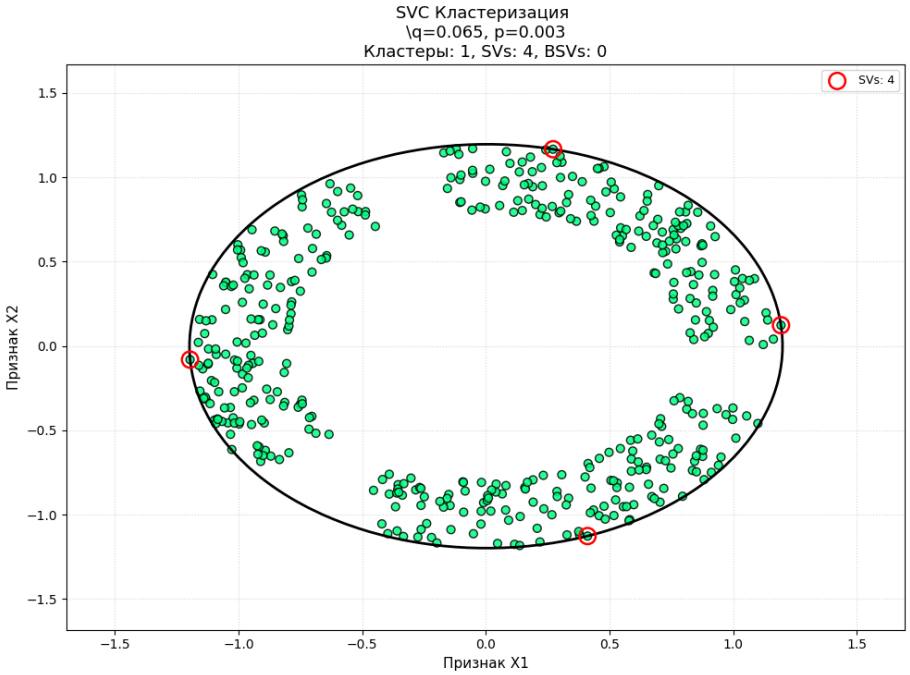
2.4. Подготовка модельных данных: луны

Попробуем поработать с другой выборкой. Сгенерируем набор из трёх лун (дуг) с небольшими зазорами между.

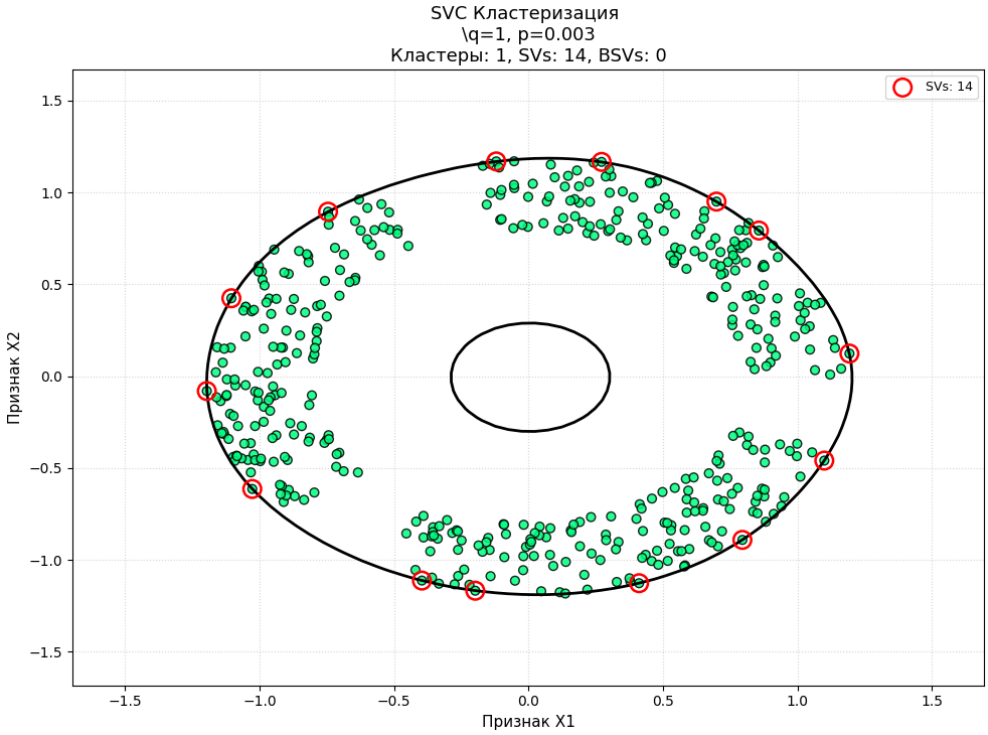


2.5. Луны: подбор значений гиперпараметров SVC-алгоритма

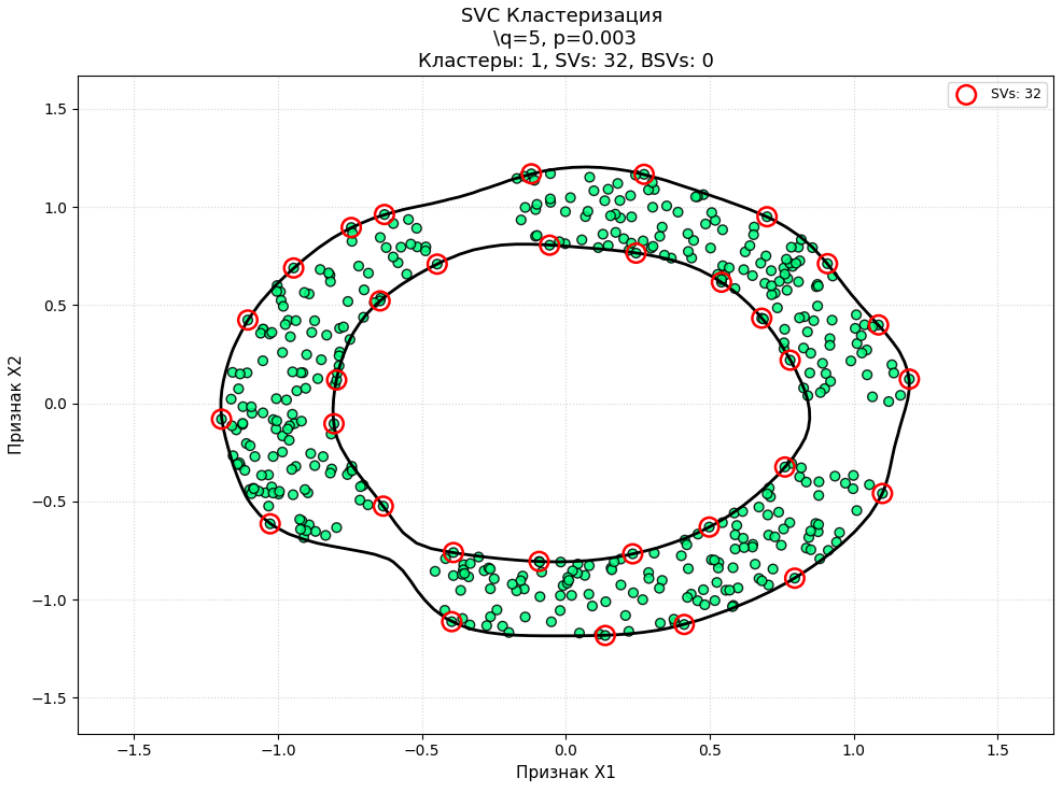
По аналогии с процедурой подбора гиперпараметров для предыдущей выборки, зафиксируем стартовые значения p=0.003 и q= 0.065. В результате так же видим единый кластер с четырмя SV.



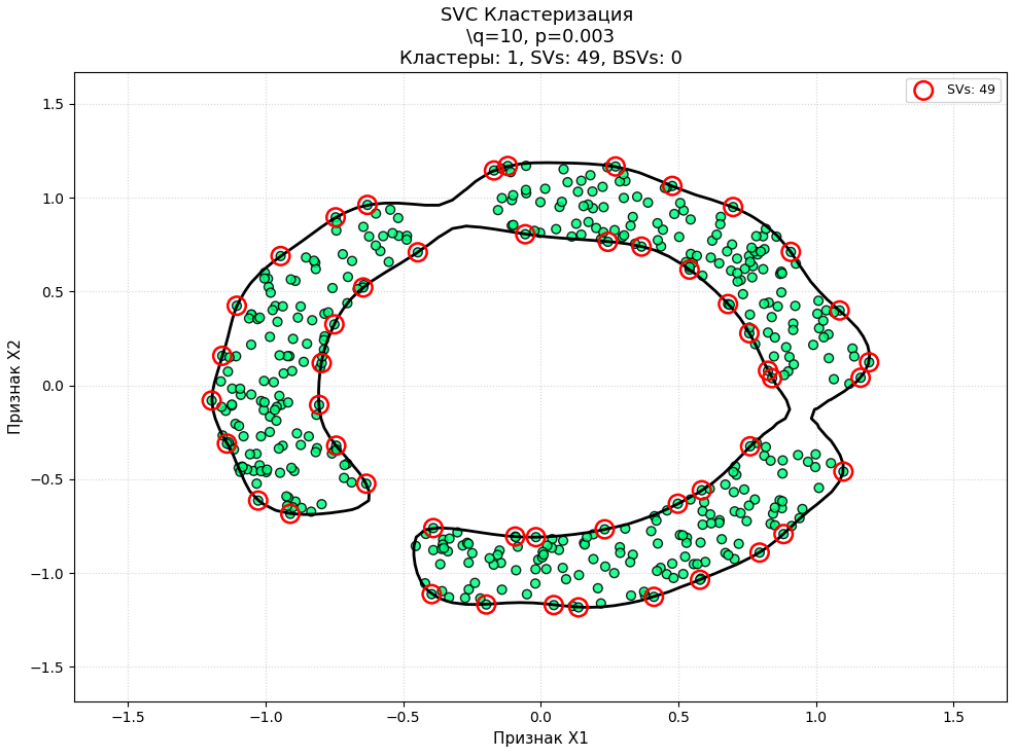
Попробуем постепенное увеличение q, переберем значения: 1, 5, 10, 15. При q=1 граница распадается на две замкнутые кривые, внешнюю и внутреннюю, в форму кольца, обтекая каждый кластер по дуге, но при этом все точки по-прежнему оказываются в единой связной компоненте (1 кластер). Число SV выросло до 14.



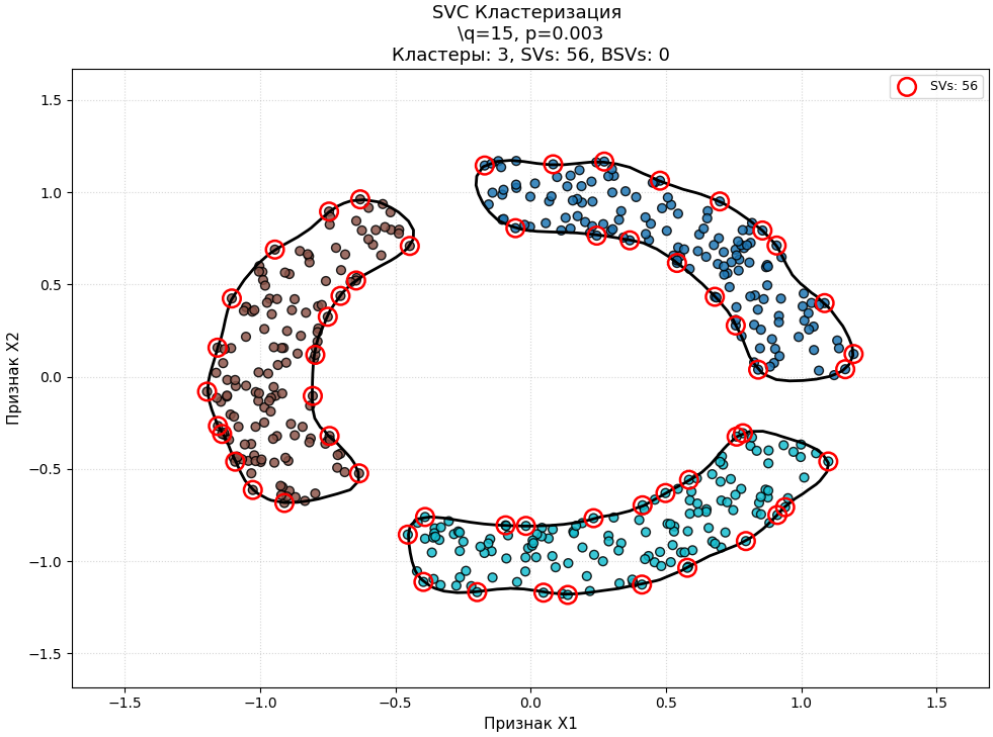
По мере роста q граница всё плотнее обходит луны, формируя при q=5 сначала одну большую связную область с внутренним «пустотелым» кольцом



При q=10 мы по-прежнему видим 1 кластер, но с всё более изрезанным контуром и растущим числом SV: с 32 до 49)



Наконец при q=15 контур распадается на три отдельные компоненты, которые соответствует трём кластерам-лунам, SV при этом 56, BSV - 0. Заметим, что для расщепления кольца на отдельные группы понадобилось достаточно большое значение q, увеличивать p при этом не пришлось.



Результаты продемонстрированы в таблице:

q	p	Количество кластеров	SVs	BSVs
0.065	0.003	1	4	0
1.000	0.003	1	14	0
5.000	0.003	1	32	0
10.000	0.003	1	49	0
15.000	0.003	3	56	0

ВЫВОДЫ

В работе был реализован SV-алгоритм кластеризации. Первым делом проверка алгоритма была осуществлена на трёх гауссовских облаках (Gaussian blobs), затем были сгенерированы три сегмента лун.

В процессе реализации алгоритма важную роль имеет подбор значений гиперпараметров – p и q. В случаях реализаций SV-алгоритма как на «гауссовских облаках», так и «лунах» показала, что при низких значениях (p=0.003 и q= 0.065) модель ведёт себя очень «мягко»: граница сглажена, в единый кластер попадает вся исследуемая масса объектов, число опорных векторов при этом минимально, связанных опорных векторов (BSV) вовсе нет.

Постепенное увеличение q приводит к росту числа опорных векторов и появлению дополнительных кластеров: контур SVC становится всё более детализированным и способным разделять структуры сложной геометрии.