1. Постановка задачи

В рамках данной лабораторной работы необходимо реализовать метод sv-кластеризации и проверить его работу на хорошо и плохо отделимых модельных данных.

План работы:

1. Подготовка модельных данных
2. Реализация SVC-алгоритма
3. Кластеризация с разными значениями гиперпараметров. Оценка качества и выбор оптимальных значений гиперпараметров.
4. Графическая иллюстрация результатов, таблицы.

2. Алгоритм SVC-кластеризации

Алгоритм SVC-кластеризации включает в себя следующие шаги:

1. Построение ядерной матрицы: $K = \{k_{ij}\}, i, j = 1, \dots, n$

$$k_{ij} = k(x_i, x_j) = exp(-q||x_i - x_j||^2)$$

2. Решение задачи квадратичного программирования. Необходимо найти множители Лагранжа $\mu_i, i = 1, \dots, n$. Любые множители меньше 10^{-3} будем считать нулем.

$$\max_{\mu_i} W$$

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

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

$$0 \leq \mu_i \leq C, i = 1, \dots, n$$

3. Классификация векторов:

- Связанные опорные вектора (BSVs): $\mu_i = C$ лежат вне гиперсферы
- Опорные вектора (SVs): $0 < \mu_i < C$ лежат на поверхности гиперсферы.
- Другие векторы (OVs): $\mu_i = 0$ - внутри гиперсферы или на поверхности

1. Расстояние от центра гиперсферы до вектора x и радиусы гиперсфер:

$$r^2(x) = k(x, x) - 2 \sum_{i=1}^{n} \mu_i k(x_i, x) + \sum_{i=1}^{n} \sum_{j=1}^{n} \mu_i \mu_j k(x_i, x_j)$$

$$R = r(x_i), x_i \in SVs$$

2. Маркировка кластеров. Необходимо построить матрицу смежности A.

$$A_{ij} = \begin{cases} 1 & \text{, если } r(y) \leq R, \forall y \in [x_i, x_j] \\ 0 & \text{, иначе} \end{cases}$$

3. Находим кластеры как связанные компоненты графа, индуцированного матрицей A. Связанные опорные векторы не кластеризуются и впоследствии относятся к кластерам с ближайшим к ним центром.

3. Критерии оценки качества кластеризации

Для оценки воспользуемся критерием Калининского-Харабаша:

$$CH(C) = \frac{N-K}{K-1} \frac{\sum\limits_{C_k \in C} |C_k| \rho(\overline{C_k}, \overline{X})}{\sum\limits_{C_k \in C} \sum\limits_{x_i \in C_k} \rho(x_i, \overline{C_k})}$$

где N - размер кластеризуемой выборки, K - количество кластеров, $|C_k|$ - количество элементов в кластере C_k, $\overline{C_k}$ - центр кластера C_k

$$\overline{C_k} = \frac{\sum\limits_{x_i \in C_k} x_i}{|C_k|}$$

$\overline{X}$ - выборочное среднее всех исходных данных

$$\overline{X} = \frac{1}{N} \sum_{x_i \in X} x_i$$

Чем больше значение критерия, тем лучше качество кластеризации.

4. Код

Код SVC алгоритма и оценки качества:

In [1]:
```
from matplotlib import pyplot
import pandas as pd
import numpy as np
import cvxopt


class SVC:
    zero_mu = 1e-3

    @staticmethod
    def init_clustering(sample, p, q):
        svc = SVC(sample, p, q)
```

```
        svc.__find_kernel_matrix()
        svc.__find_mu()
        svc.__find_segment_matrix()
        svc.__find_clusters()
        return svc

    def show_plot(self, max_clusters_count=10):
        labels = np.zeros(self.sample.shape[0])
        for i in self.clusters.keys():
            for j in self.clusters[i]:
                labels[j] = int(i)
        if len(self.clusters) >= max_clusters_count:
            print("Number of clusters is more or equals 10, so no graphic")
            return
        df = pd.DataFrame(dict(x=self.sample[:, 0], y=self.sample[:, 1], lab
        colors = {1: 'r', 2: 'b', 3: 'g', 4: 'c', 5: 'm', 6: 'y', 7: 'k', 8:
        fig, ax = pyplot.subplots()
        grouped = df.groupby('label')
        for key, group in grouped:
            group.plot(ax=ax, kind='scatter', x='x', y='y', label=key, color

        # Помечаем звездочками опорные вектора
        svs_val_x = [self.sample[i][0] for i in self.svs]
        svs_val_y = [self.sample[i][1] for i in self.svs]
        ax.scatter(x=svs_val_x, y=svs_val_y, marker='*', color='black')

        # Помечаем крестами связанные опорные вектора
        svs_val_x = [self.sample[i][0] for i in self.bsvs]
        svs_val_y = [self.sample[i][1] for i in self.bsvs]
        ax.scatter(x=svs_val_x, y=svs_val_y, marker='s', color='slategray')

        pyplot.show()

    def calinski_harabasz(self):
        # K - количество кластеров
        k = len(self.clusters)
        if k == 1:
            return '-'
        if k == self.N:
            return 0

        # X - выборочное среднее всех исходных данных
        x_mean = sum(self.sample) / self.N
        scale = (self.N - k) / (k - 1)

        numerator = 0
        denominator = 0

        for cluster_num in self.clusters:
            cluster = self.clusters[cluster_num]
            # |C_k| - количество элементов в кластере
            cluster_len = len(cluster)

            cluster_sum = 0
            for id in cluster:
                cluster_sum += self.sample[id]

            # \overline{C_k} - центр кластера C_k
            cluster_center = cluster_sum / cluster_len
            numerator += cluster_len * SVC.rho(cluster_center, x_mean)

            # Считаем сумму расстояний до центра кластера
            cluster_distance_sum = 0
            for id in cluster:
```

```
                cluster_distance_sum += SVC.rho(cluster_center, self.sample[

            denominator += cluster_distance_sum

        return scale * numerator / denominator

    # Don't use constructor, use init_clustering
    # sample - выборка, которую кластеризуем
    def __init__(self, sample, p, q):
        self.sample = sample
        self.N = len(sample)
        self.p = p
        self.q = q
        self.C = 1 / (self.N * p)

        # Ядерная матрица
        self.km = None
        # Вектор множителей Лагранжа \mu
        self.mu = None
        # Матрица смежности A
        self.segment_matrix = None
        self.clusters = None
        self.svs = None
        self.bsvs = None

    # Инициализация ядерной матрицы
    def __find_kernel_matrix(self):
        self.km = np.zeros((self.N, self.N))
        for i in range(self.N):
            for j in range(self.N):
                self.km[i, j] = self.__kernel(self.sample[i], self.sample[j]

    # Инициализируем множители Лагранжа \mu_i
    def __find_mu(self):
        P = cvxopt.matrix(self.km)
        q = cvxopt.matrix(self.km.diagonal().reshape(self.N, 1))
        G1 = cvxopt.spmatrix(-1, range(self.N), range(self.N))
        G2 = cvxopt.spmatrix(1, range(self.N), range(self.N))
        G = cvxopt.sparse([G1, G2])
        h1 = np.zeros(self.N)
        h2 = np.full(self.N, self.C)
        h = cvxopt.matrix(np.concatenate((h1, h2), axis=0), (2 * self.N, 1))
        A = cvxopt.matrix(1.0, (1, self.N))
        b = cvxopt.matrix(1.0)
        cvxopt.solvers.options['show_progress'] = False
        sol = cvxopt.solvers.qp(P, q, G, h, A, b)
        self.mu = np.array(sol['x'])

    def __find_segment_matrix(self):
        svs_tmp = np.array(self.C > self.mu + SVC.zero_mu) * np.array(self.m
        self.svs = np.where(svs_tmp == True)[0]
        bsvs_tmp = np.array(np.isclose(self.mu, self.C, atol=SVC.zero_mu))
        self.bsvs = np.where(bsvs_tmp == True)[0]
        r = np.mean([self.__r2_func(self.sample[i]) for i in self.svs[:5]])
        self.segment_matrix = np.zeros((self.N, self.N))
        for i in range(self.N):
            if i not in self.bsvs:
                for j in range(i, self.N):
                    if j not in self.bsvs:
                        self.segment_matrix[i, j] = self.segment_matrix[j, i


    def __find_clusters(self):
        ids = list(range(self.N))
```

```
        self.clusters = {}
        num_clusters = 0
        while ids:
            num_clusters += 1
            self.clusters[num_clusters] = []
            curr_id = ids.pop(0)
            queue = [curr_id]
            while queue:
                cid = queue.pop(0)
                for i in ids:
                    if self.segment_matrix[i, cid]:
                        queue.append(i)
                        ids.remove(i)
                self.clusters[num_clusters].append(cid)

    def __r2_func(self, x):
        return 1 - 2 * sum(
            self.mu[i] * self.__kernel(self.sample[i], x) for i in range(sel
        )

    def __kernel(self, x1, x2):
        return np.exp(-self.q * sum((x1 - x2) ** 2))

    # Вычисление элемента матрицы смежности A_ij
    def __segment(self, x1, x2, r, n=10):
        for i in range(n):
            x = x1 + (x2 - x1) * i / (n + 1)
            if self.__r2_func(x) > r:
                return False
        return True

    # Расстояние между 2мя точками
    @staticmethod
    def rho(x1, x2):
        return np.linalg.norm(x1 - x2)
```

Код для формирования хорошо и плохо отделимых данных:

In [2]:

```
from scipy.stats import multivariate_normal


def get_sample(n, s):
    m1 = np.array([2, 1])
    m2 = np.array([0, 13])
    m3 = np.array([15, 8])
    size = int(n / 3)
    cluster1 = multivariate_normal(mean=m1, cov=s).rvs(size)
    cluster2 = multivariate_normal(mean=m2, cov=s).rvs(size)
    if n % 3 != 0:
        size += 1
    cluster3 = multivariate_normal(mean=m3, cov=s).rvs(size)

    pyplot.scatter(x=cluster1[:, 0], y=cluster1[:, 1], color='r')
    pyplot.scatter(x=cluster2[:, 0], y=cluster2[:, 1], color='g')
    pyplot.scatter(x=cluster3[:, 0], y=cluster3[:, 1], color='b')
    pyplot.show()

    return np.vstack([cluster1, cluster2, cluster3])


n = 100
s_good = np.array([[2, 0],
```

```
                    [0, 2]])
s_bad = s_good * 3
```

Код для исследования качества кластеризации и построения итоговой таблицы:

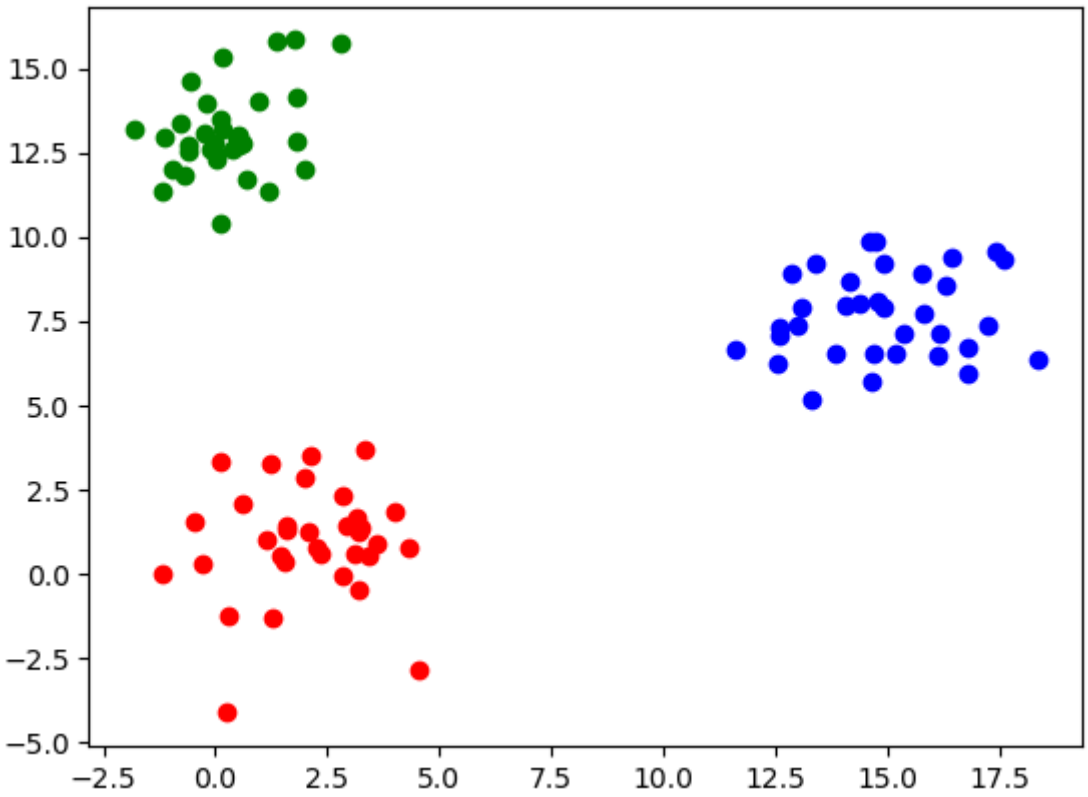
In [3]:
```
def research(sample, max_cluster_count=10):
    max = np.max([sum((sample[i] - sample[j]) ** 2) for i in range(len(sampl
    q1 = 1 / max
    print(f"q1 = {q1}")

    pq_set = (
        (0.01, q1),
        (0.01, 0.1),
        (0.01, 0.5),
        (0.1, 0.1),
        (0.5, 0.1),
        (0.9, 0.1),
        (1, 0.1)
    )

    table = pd.DataFrame(columns=['p', 'q', 'clusters count', 'SVs', 'BSVs',
    for pq in pq_set:
        svc = SVC.init_clustering(sample, p=pq[0], q=pq[1])
        ch = svc.calinski_harabasz()

        print("p, q, clusters count, SVs, BSVs, CH")
        row = [pq[0], pq[1], len(svc.clusters), len(svc.svs), len(svc.bsvs),
        print(row)
        table.loc[len(table)] = row

        svc.show_plot(max_cluster_count)

    return table
```

In [11]:
```
%%html
<style>
.output_wrapper .output {
  overflow-y: visible;
  height: fit-content;
}
</style>
```

5. Хорошо отделимые данные

Посмотрим как работает наш алгоритм на хорошо отделимых данных, возьмем 100 точек в 3х разных кластерах

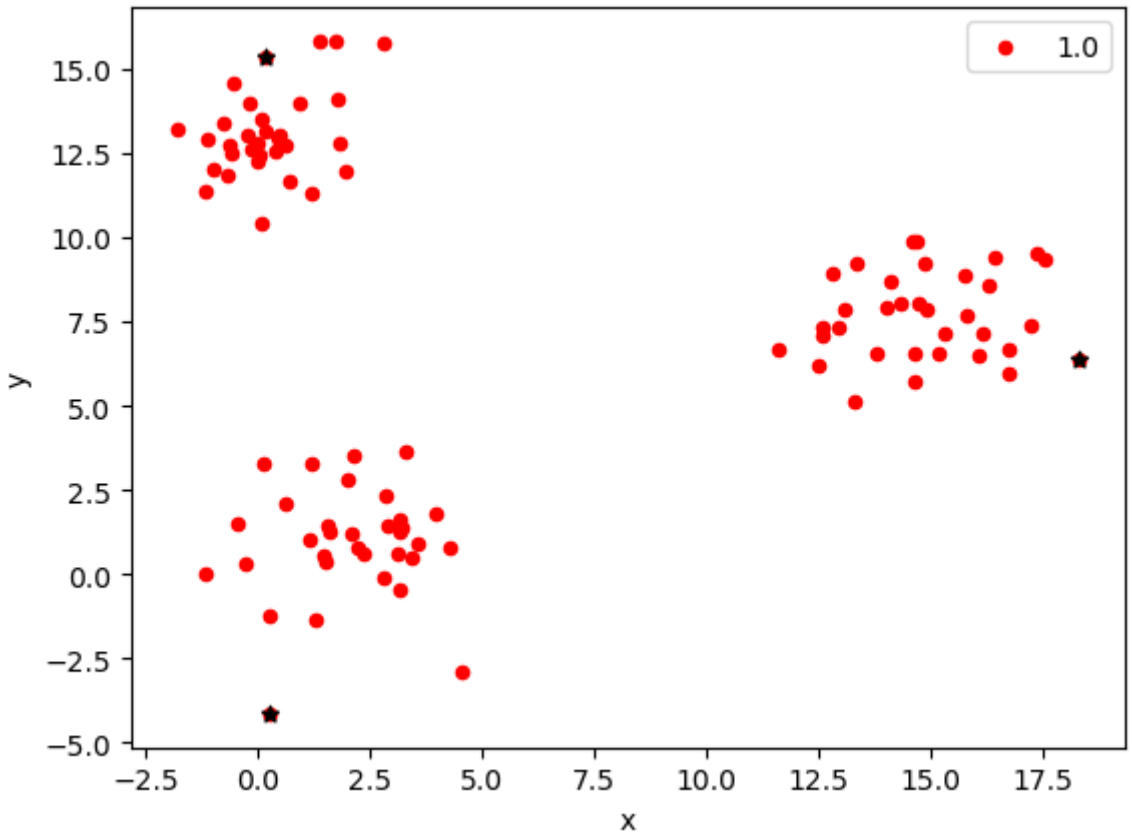
In [4]:
```
good_sample = get_sample(100, s_good)
```



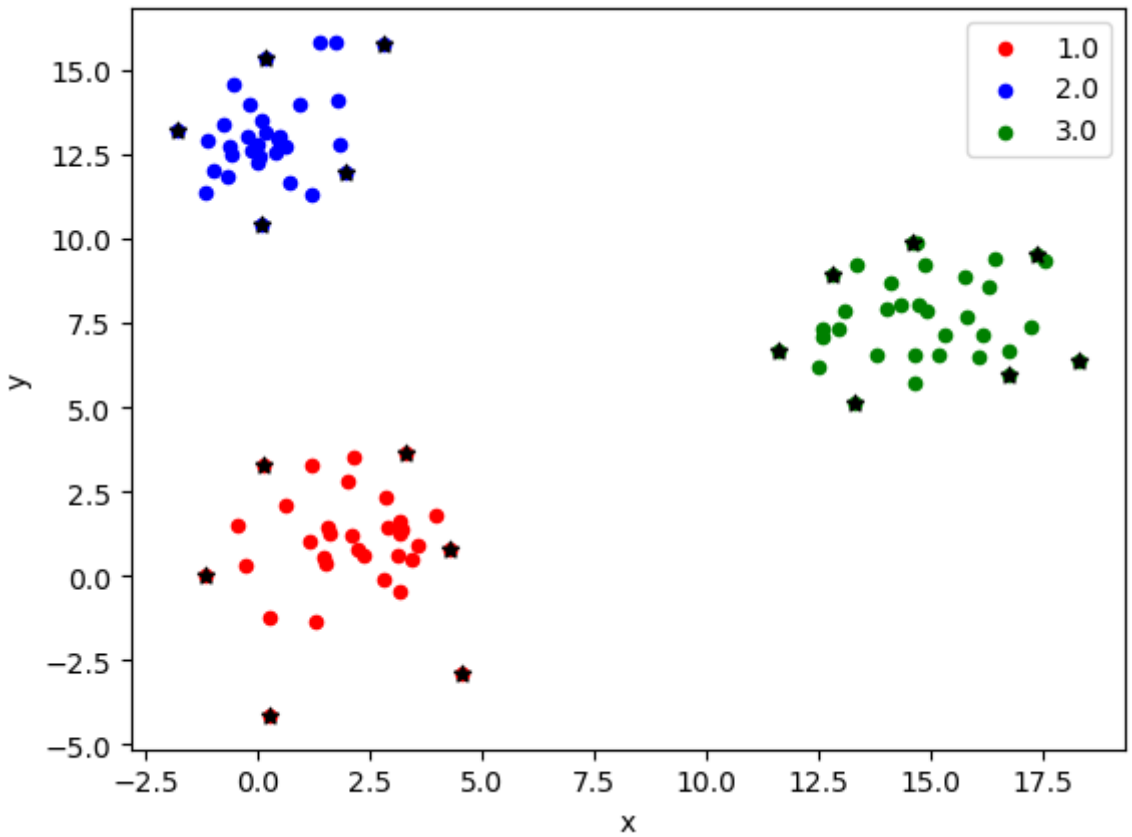
Построим графики полученной алгоритмом кластеризации для разных параметров p и q и сравним их в итоговой таблице. Графики, на которых больше 10 кластеров, выводить не будем. Звездочками отвечены опорные вектора, а квадратами связанные опорные вектора.

In [5]:
```
good_sample_table = research(good_sample)
```

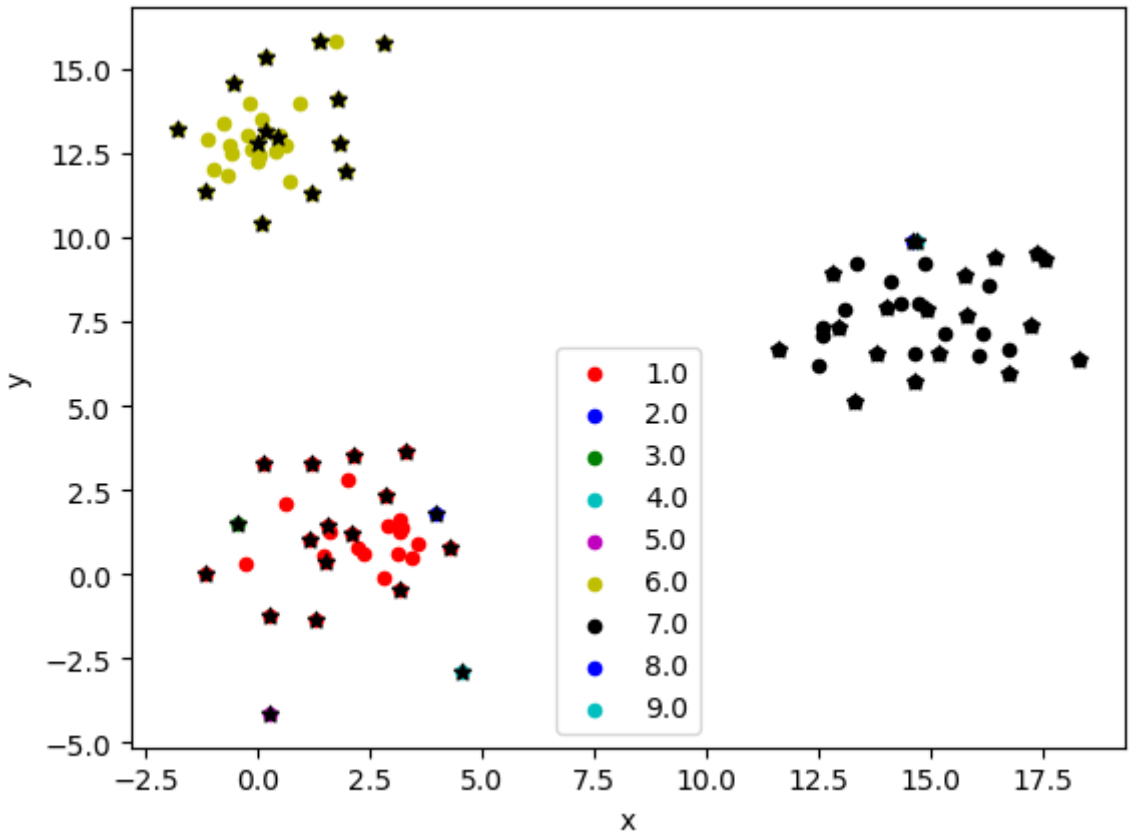
```
q1 = 0.002075480485749144
p, q, clusters count, SVs, BSVs, CH
[0.01, 0.002075480485749144, 1, 3, 0, '-']
```



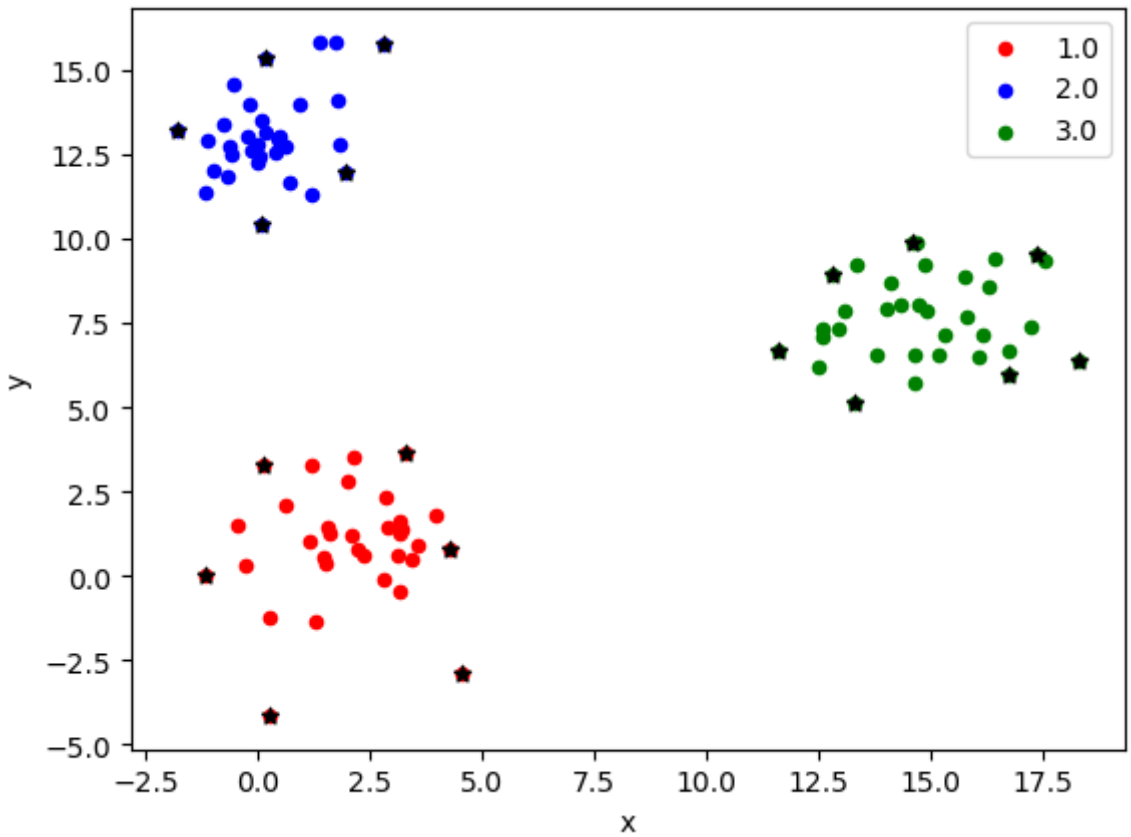

```
p, q, clusters count, SVs, BSVs, CH
[0.01, 0.1, 3, 18, 0, 232.4020878646505]
```




```
p, q, clusters count, SVs, BSVs, CH
[0.01, 0.5, 9, 51, 0, 61.621402731817525]
```




```
p, q, clusters count, SVs, BSVs, CH
[0.1, 0.1, 3, 18, 0, 232.4020878646505]
```



```
p, q, clusters count, SVs, BSVs, CH
[0.5, 0.1, 50, 6, 47, 16.921356880896514]
Number of clusters is more or equals 10, so no graphic
p, q, clusters count, SVs, BSVs, CH
[0.9, 0.1, 92, 3, 88, 24.57782927140867]
Number of clusters is more or equals 10, so no graphic
p, q, clusters count, SVs, BSVs, CH
[1, 0.1, 100, 0, 100, 0]
Number of clusters is more or equals 10, so no graphic
```

```
/Users/lev.saskov/opt/anaconda3/lib/python3.9/site-packages/numpy/core/fromn
umeric.py:3440: RuntimeWarning: Mean of empty slice.
  return _methods._mean(a, axis=axis, dtype=dtype,
/Users/lev.saskov/opt/anaconda3/lib/python3.9/site-packages/numpy/core/_meth
ods.py:189: RuntimeWarning: invalid value encountered in double_scalars
  ret = ret.dtype.type(ret / rcount)
```

Итоговая таблица:

In [6]:
```
good_sample_table
```

Out[6]:

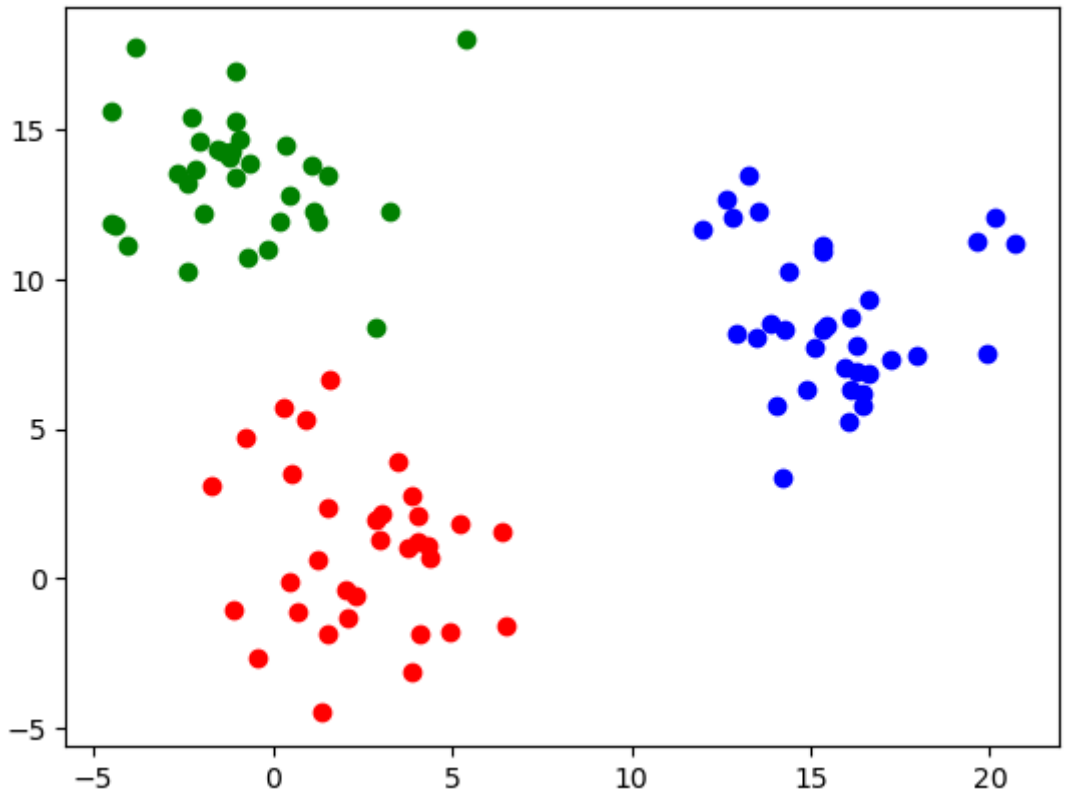
	p	q	clusters count	SVs	BSVs	CH
0	0.01	0.002075	1.0	3.0	0.0	-
1	0.01	0.100000	3.0	18.0	0.0	232.402088
2	0.01	0.500000	9.0	51.0	0.0	61.621403
3	0.10	0.100000	3.0	18.0	0.0	232.402088
4	0.50	0.100000	50.0	6.0	47.0	16.921357
5	0.90	0.100000	92.0	3.0	88.0	24.577829
6	1.00	0.100000	100.0	0.0	100.0	0.0

Видим, что лучшее разбиение получилось при параметрах p и q равных 0.01 и 0.1.

6. Плохо отделимые данные

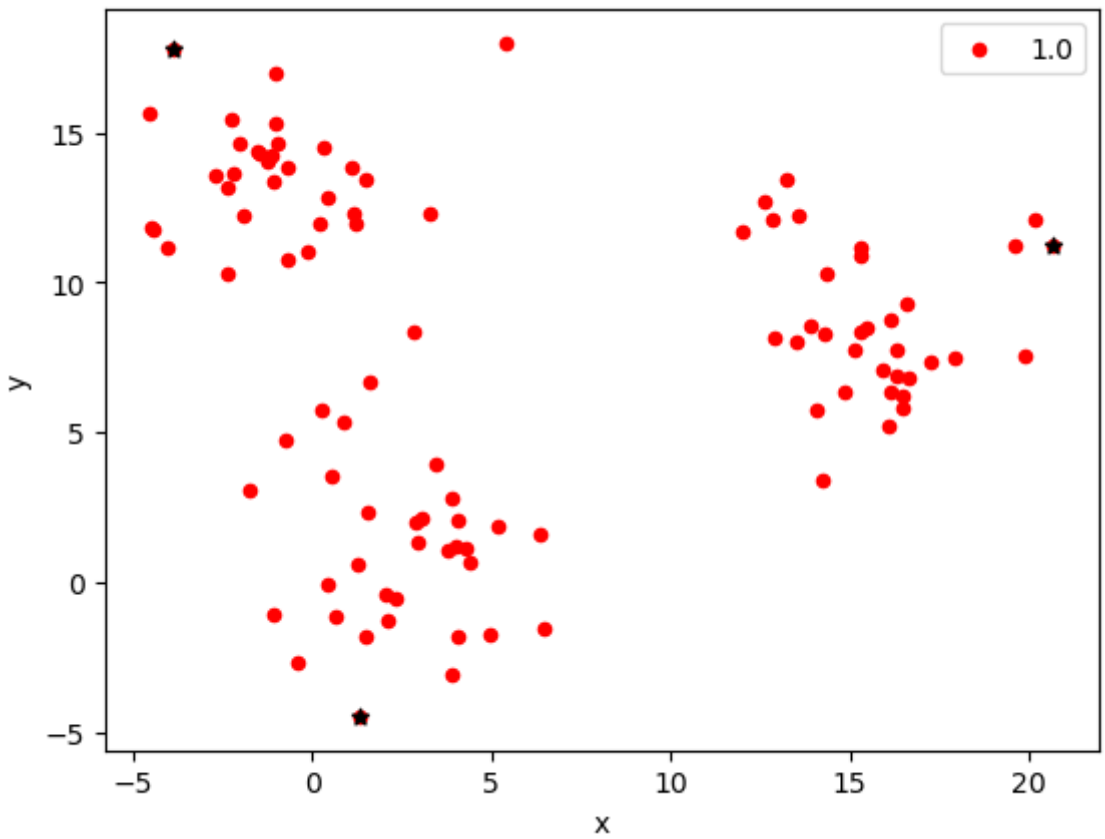
Посмотрим как ведет себя алгоритм при работе с плохо отделимыми данными.
Графики построим только для тех случаев, где меньше 20 кластеров.

In [7]:
```
bad_sample = get_sample(100, s_bad)
```

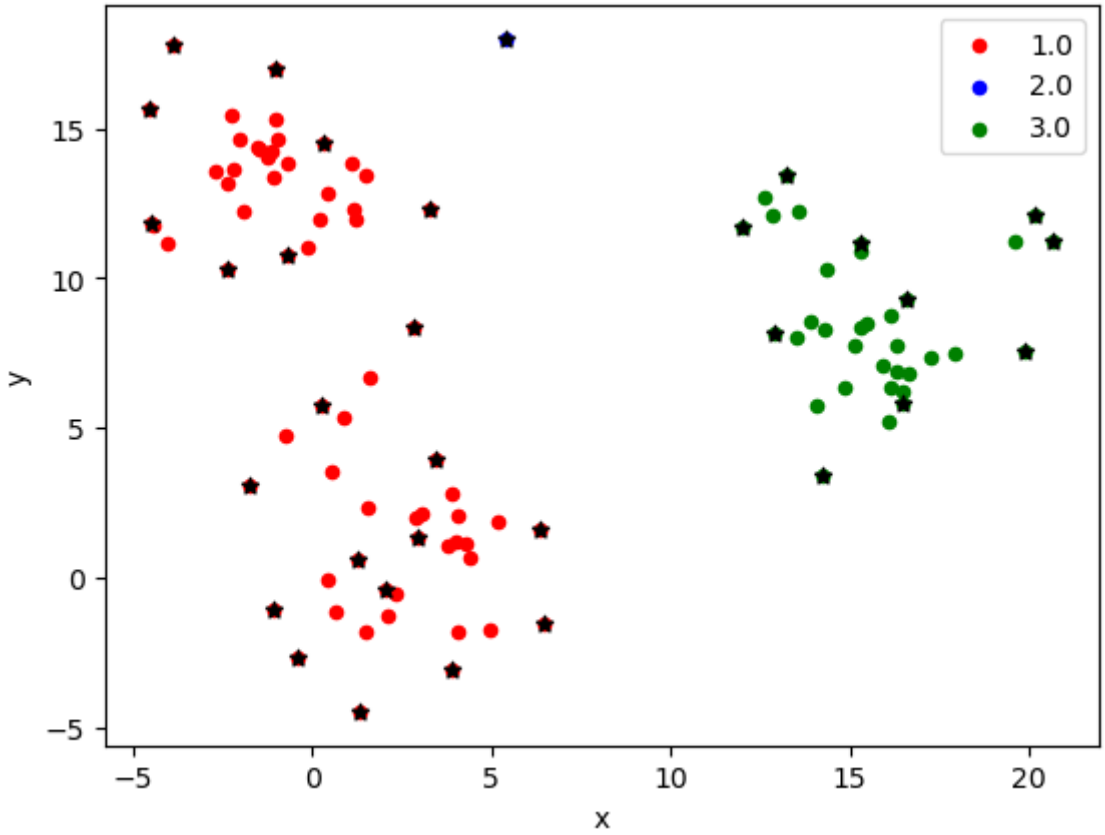


In [8]:
```
bad_sample_table = research(bad_sample, max_cluster_count=20)
```

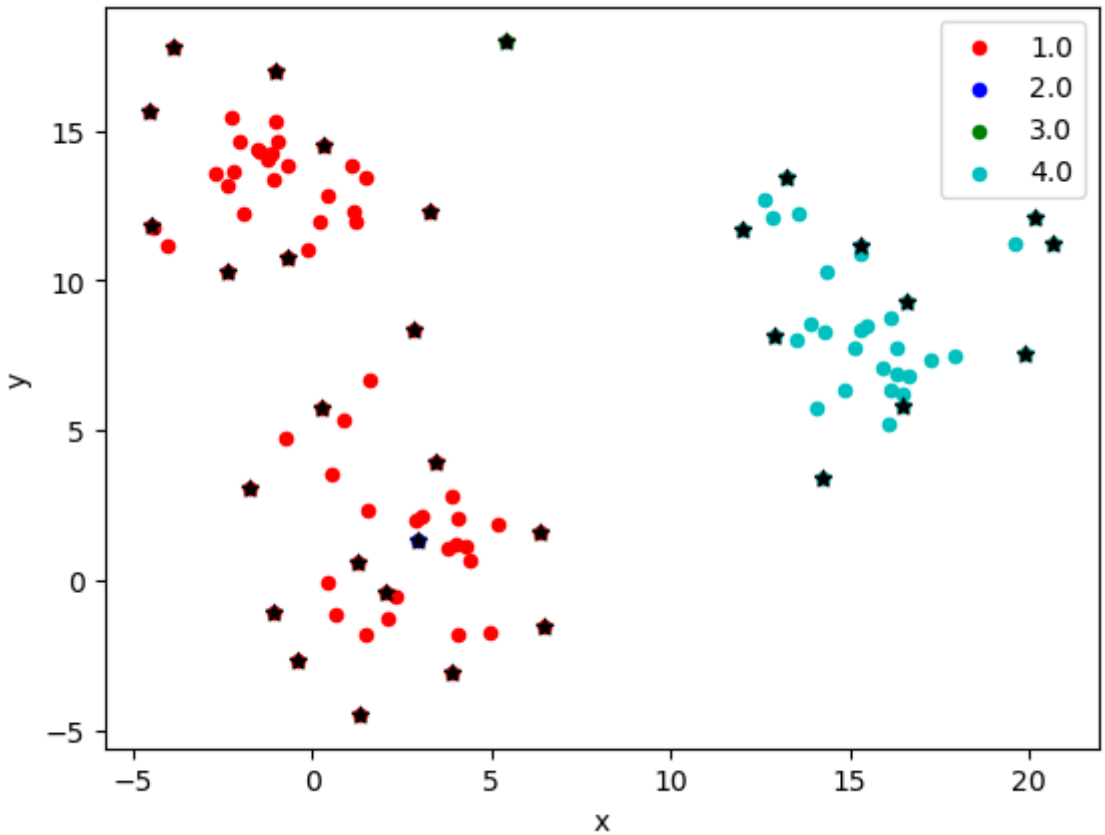
```
q1 = 0.001495030607775564
p, q, clusters count, SVs, BSVs, CH
[0.01, 0.001495030607775564, 1, 3, 0, '-']
```




```
p, q, clusters count, SVs, BSVs, CH
[0.01, 0.1, 3, 32, 0, 61.89570445278728]
```




```
p, q, clusters count, SVs, BSVs, CH
[0.01, 0.5, 25, 69, 0, 24.40354098663789]
Number of clusters is more or equals 10, so no graphic
p, q, clusters count, SVs, BSVs, CH
[0.1, 0.1, 4, 32, 0, 41.51948266036185]
```



```
p, q, clusters count, SVs, BSVs, CH
[0.5, 0.1, 47, 14, 43, 10.509960278301222]
Number of clusters is more or equals 10, so no graphic
p, q, clusters count, SVs, BSVs, CH
[0.9, 0.1, 92, 3, 89, 14.882591317371032]
Number of clusters is more or equals 10, so no graphic
p, q, clusters count, SVs, BSVs, CH
[1, 0.1, 100, 0, 100, 0]
Number of clusters is more or equals 10, so no graphic
/Users/lev.saskov/opt/anaconda3/lib/python3.9/site-packages/numpy/core/fromn
umeric.py:3440: RuntimeWarning: Mean of empty slice.
  return _methods._mean(a, axis=axis, dtype=dtype,
/Users/lev.saskov/opt/anaconda3/lib/python3.9/site-packages/numpy/core/_meth
ods.py:189: RuntimeWarning: invalid value encountered in double_scalars
  ret = ret.dtype.type(ret / rcount)
```

Итоговая таблица:

In [9]:
```
bad_sample_table
```

Out[9]:

	p	q	clusters count	SVs	BSVs	CH
0	0.01	0.001495	1.0	3.0	0.0	-
1	0.01	0.100000	3.0	32.0	0.0	61.895704
2	0.01	0.500000	25.0	69.0	0.0	24.403541
3	0.10	0.100000	4.0	32.0	0.0	41.519483
4	0.50	0.100000	47.0	14.0	43.0	10.50996
5	0.90	0.100000	92.0	3.0	89.0	14.882591
6	1.00	0.100000	100.0	0.0	100.0	0.0

7. Выводы

- Был рассмотрен и реализован SVC-алгоритм кластеризации
- Протестировали работу алгоритма на хорошо и плохо отделимых данных
- Сравнили качество работы алгоритма с разными гиперпараметрами с помощью критерия Калининского-Харабаша
- На хорошо отделимых данных алгоритм работает лучше
- Для хорошо отделимых данных оптимальными гиперпараметрами, дающими наилучший результат стали p = 0.01, 0.1 и q = 0.1
- Для плохо отделимых данных наилучшими гиперпарметрами стали p = 0.01; q = 0.1
- Начиная с какого-то момента, при увеличении гиперпараметров падает качество кластеризации и растет количество связанных опорных векторов
- Также было замечено, что при наличии связанных опорных векторов кластеризация не дает хорошие результаты, такие результаты кластеризации могут быть связаны с довольно сложной границей между кластерами

In [ ]:
```
```