Лабораторная работа №16

Кластеризация

Как мы уже говорили выше, кластеризация (**clustering**) является задачей разбиения набора данных на группы, называемые кластерами. Цель – разделить данные таким образом, чтобы точки, находящие в одном и том же кластере, были очень схожи друг с другом, а точки, находящиеся в разных кластерах, отличались друг от друга. Как и алгоритмы классификации, алгоритмы кластеризации присваивают (или прогнозируют) каждой точке данных номер кластера, которому она принадлежит.

Кластеризация k-средних

Кластеризация **k-средних** – один из самых простых и наиболее часто используемых алгоритмов кластеризации. Сначала выбирается число кластеров k. После выбора значения k алгоритм k-средних отбирает точки, которые будут представлять центры кластеров (**cluster centers**). Затем для каждой точки данных вычисляется его евклидово расстояние до каждого центра кластера. Каждая точка назначается ближайшему центру кластера. Алгоритм вычисляет центроиды (**centroids**) – центры тяжести кластеров. Каждый центроид – это вектор, элементы которого представляют собой средние значения характеристик, вычисленные по всем точкам кластера. Центр кластера смещается в его центроид. Точки заново назначаются ближайшему центру кластера. Этапы изменения центров кластеров и переназначения точек итеративно повторяются до тех пор, пока границы кластеров и расположение центроидов не перестанут изменяться, т.е. на каждой итерации в каждый кластер будут попадать одни и те же точки данных. Следующий пример (рис. 16.1) иллюстрирует работу алгоритма на синтетическом наборе данных

```
mglearn.plots.plot_kmeans_algorithm()
plt.show()
```

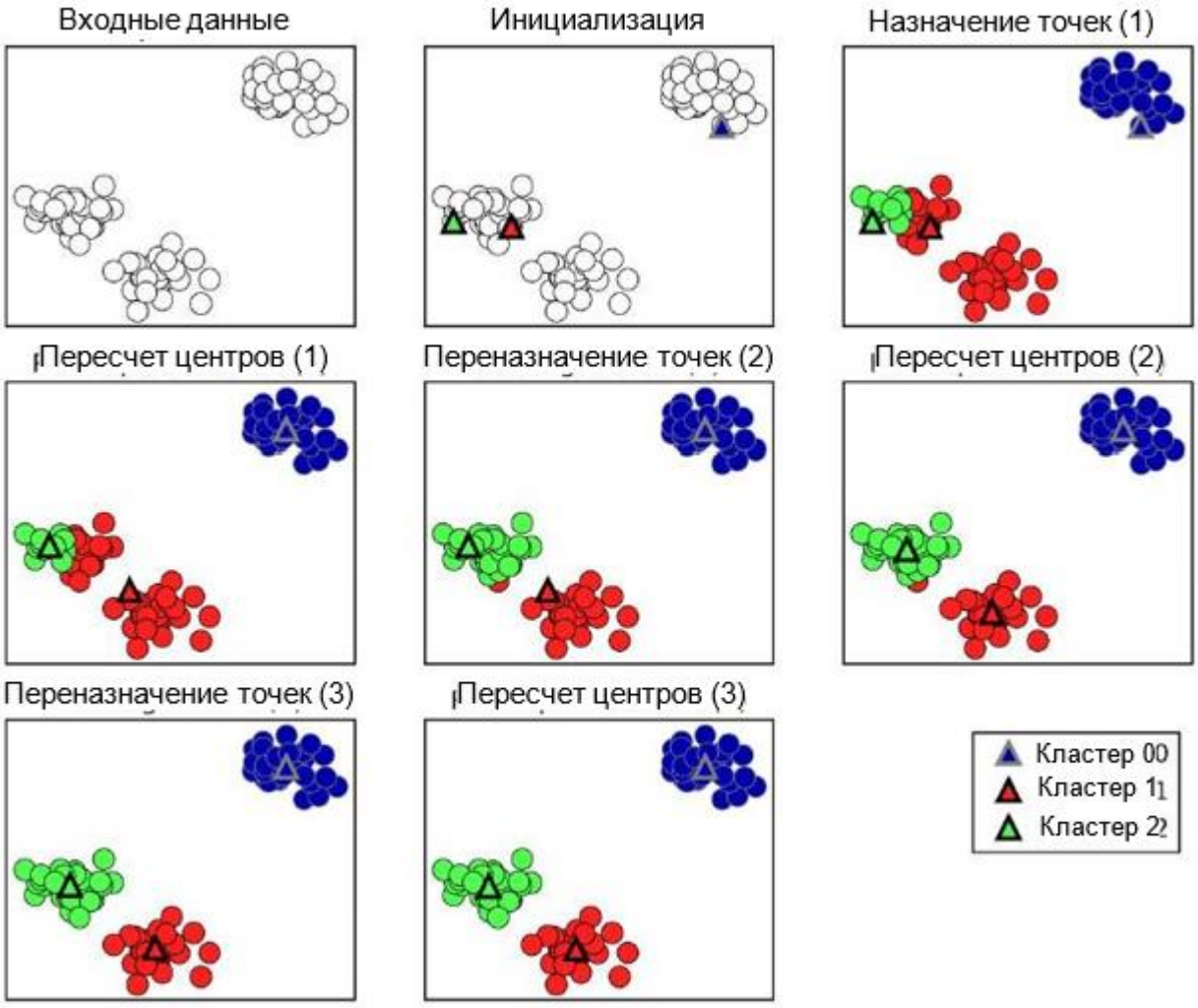

Рис. 16.1 Исходные данные и этапы алгоритма k-средних

Центры кластеров представлены в виде треугольников, в то время как точки данных отображаются в виде окружностей. Цвета указывают принадлежность к кластеру. Мы указали, что ищем три кластера, поэтому алгоритм был инициализирован с помощью случайного выбора трех точек данных в качестве центров кластеров (см. «Инициализация»). Затем запускается итерационный алгоритм. Во-первых, каждая точка данных назначается ближайшему центру кластера (см. «Назначение точек (1)»). Затем центры кластеров переносятся в центры тяжести кластеров (см. «Пересчет центров (1)»). Затем процесс повторяется еще два раза. После третьей итерации принадлежность точек кластерным центрам не изменилась, поэтому алгоритм останавливается.

Получив новые точки данных, алгоритм k-средних будет присваивать каждую точку данных ближайшему центру кластера. Следующий пример (рис. 16.2) показывает границы центров кластеров, процесс вычисления которых был приведен на рис. 16.1:

```
mglearn.plots.plot_kmeans_algorithm()
plt.show()
```

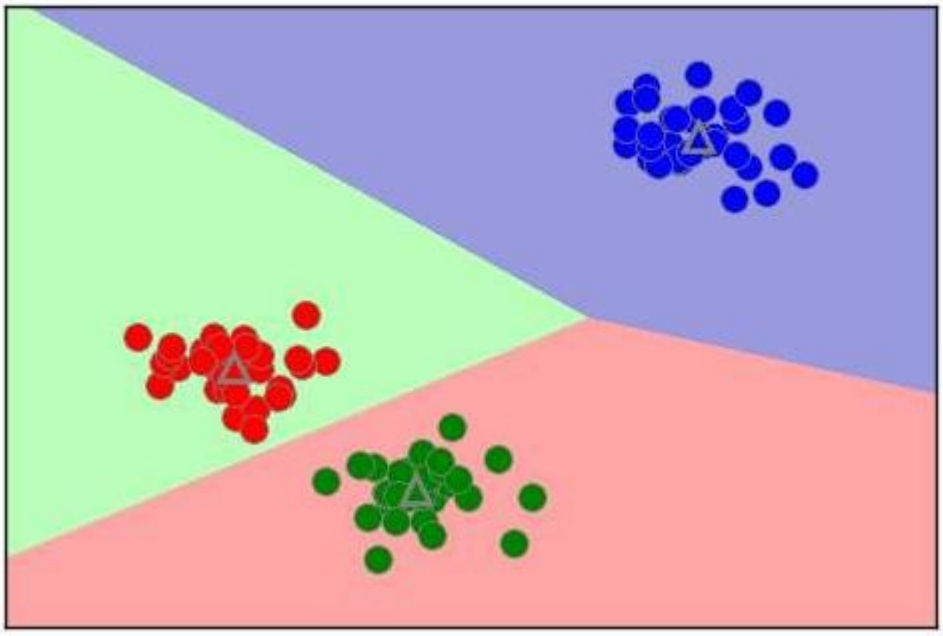

Рис. 16.2 Центры кластеров и границы кластеров, найденные с помощью алгоритма k-средних

Применить **алгоритм k-средних**, воспользовавшись библиотекой scikit-learn, довольно просто. Здесь мы применяем его к синтетическим данным, которые использовали для построения предыдущих графиков. Мы создаем экземпляр класса **KMeans** и задаем количество выделяемых кластеров. Затем мы вызываем метод **fit** и передаем ему в качестве аргумента данные:

```
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans

# генерируем синтетические двумерные данные
X, y = make_blobs(random_state=1)
kmeans = KMeans(n_clusters=3)
kmeans.fit(X)
```

Во время работы алгоритма каждой точке обучающих данных X присваивается метка кластера. Вы можете найти эти метки в атрибуте **kmeans.labels_**:

```
print("Принадлежность к кластерам:\n{}".format(kmeans.labels_))
```

Поскольку мы задали три кластера, кластеры пронумерованы от 0 до 2.

Кроме того, вы можете присвоить метки кластеров новым точкам с помощью метода **predict**. В ходе прогнозирования каждая новая точка назначается ближайшему центру кластера, но существующая модель не меняется. Запуск метода **predict** на обучающем наборе возвращает тот же самый результат, что содержится в атрибуте **labels_**:

```
print(kmeans.predict(X))
```

Вы можете увидеть, что кластеризация немного похожа на классификацию в том плане, что каждый элемент получает метку. Однако нет никаких оснований утверждать, что данная метка является истинной и поэтому сами по себе метки не несут никакого априорного смысла. Давайте вернемся к примеру с кластеризацией изображений лиц, который мы обсуждали ранее. Возможно, что кластер 3, найденный с помощью алгоритма, содержит лишь лица вашего друга. Впрочем, вы можете узнать это только после того, как взгляните на фотографии, а само число 3 является произвольным. Единственная информация, которую дает вам алгоритм, – это то, что все лица, отнесенные к кластеру 3, схожи между собой.

В случае с кластеризацией, которую мы только что построили для двумерного синтетического набора данных, это означает, что мы не должны придавать значения тому факту, что одной группе был присвоен 0, а другой – 1. Повторный запуск алгоритма может привести к совершенно иной нумерации кластеров в силу случайного характера инициализации.

Ниже приводится новый график для тех же самых данных (рис. 16.3). Центры кластеров записаны в атрибуте **cluster_centers_** и мы наносим их на график в виде треугольников:

```
mglearn.discrete_scatter(X[:, 0], X[:, 1], kmeans.labels_, markers='o')
mglearn.discrete_scatter(
    kmeans.cluster_centers_[:, 0],
    kmeans.cluster_centers_[:, 1],
    [0, 1, 2],
    markers='^',
    markeredgewidth=2
)
plt.show()
```

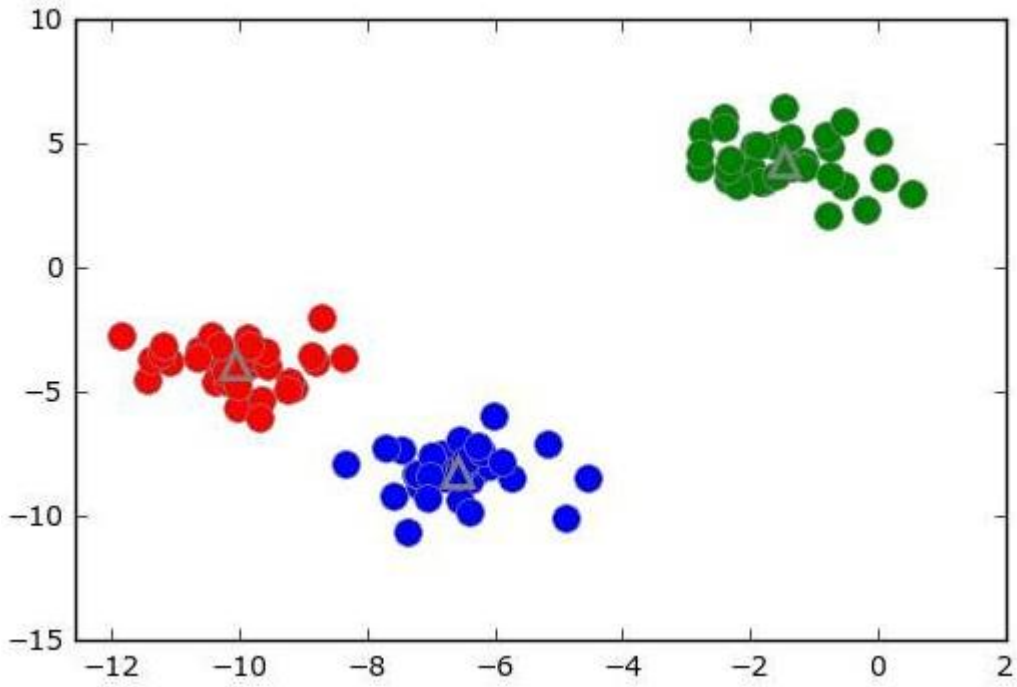

Рис. 16.3 Принадлежность к кластерам и центры кластеров, найденные с помощью алгоритма k-средних, k=3

Кроме того, мы можем увеличить или уменьшить количество центров кластеров (рис. 16.4):

```
fig, axes = plt.subplots(1, 2, figsize=(10, 5))
# использование двух центров кластеров:
kmeans = KMeans(n_clusters=2)
kmeans.fit(X)
assignments = kmeans.labels_
mglearn.discrete_scatter(X[:, 0], X[:, 1], assignments, ax=axes[0])
# использование пяти центров кластеров:
kmeans = KMeans(n_clusters=5)
kmeans.fit(X)
assignments = kmeans.labels_
mglearn.discrete_scatter(X[:, 0], X[:, 1], assignments, ax=axes[1])
plt.show()
```

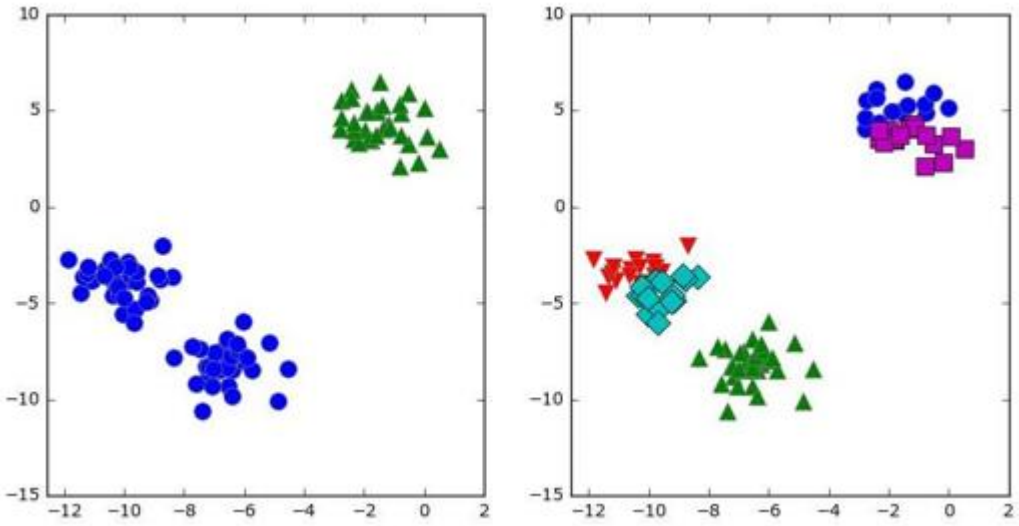

Рис. 16.4 Принадлежность к кластерам, найденная с помощью алгоритма k-средних, k=3 (слева) и k=5 (справа)

Недостатки алгоритмаk-средних

Даже если вы знаете «правильное» количество кластеров для конкретного набора данных, алгоритм **k-средних** не всегда может выделить их. Каждый кластер определяется исключительно его центром, это означает, что каждый кластер имеет выпуклую форму. В результате этого алгоритм **k-средних** может описать относительно простые формы. Кроме того, алгоритм **k-средних** предполагает, что все кластеры в определенном смысле имеют одинаковый «диаметр», он всегда проводит границу между

кластерами так, чтобы она проходила точно посередине между центрами кластеров. Это иногда может привести к неожиданным результатам, как показано на рис. 16.5:

```
X_varied, y_varied = make_blobs(n_samples=200,
                                cluster_std=[1.0, 2.5, 0.5], random_state=170)
y_pred = KMeans(n_clusters=3, random_state=0).fit_predict(X_varied)
mglearn.discrete_scatter(X_varied[:, 0], X_varied[:, 1], y_pred)
plt.legend(["кластер 0", "кластер 1", "кластер 2"], loc='best')
plt.xlabel("Признак 0")
plt.ylabel("Признак 1")
plt.show()
```

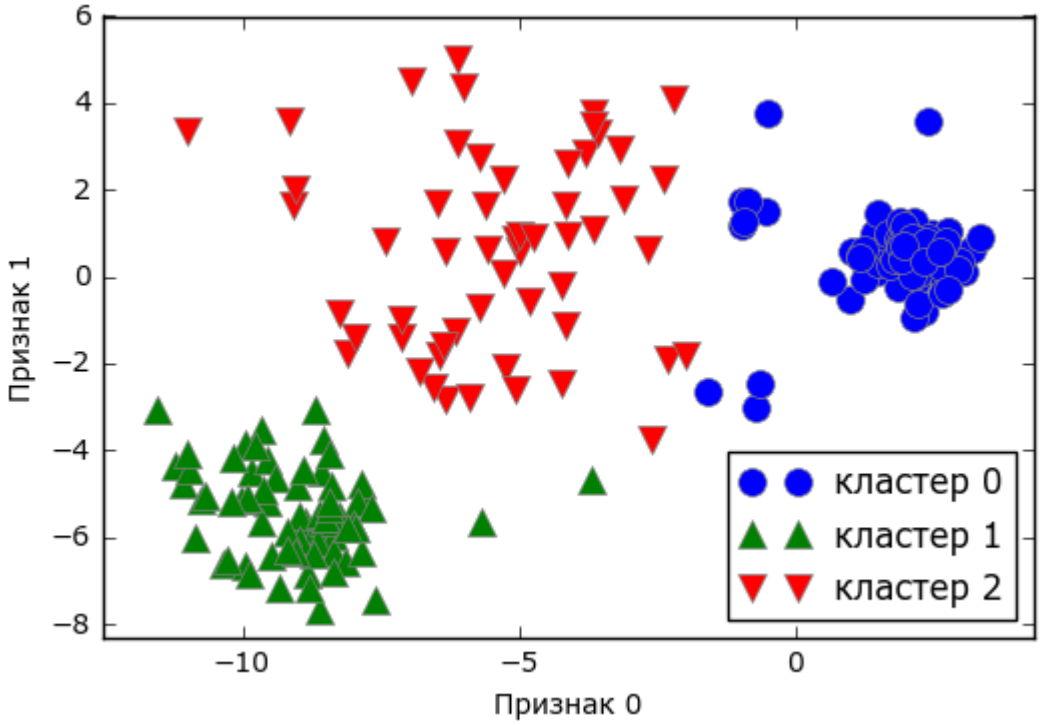

Рис. 16.5 Принадлежность к кластерам, найденная с помощью алгоритма k-средних, при этом кластеры имеют разные плотности

Можно было бы ожидать плотную область в нижнем левом углу, которая рассматривалась бы в качестве первого кластера, плотную область в верхнем правом углу в качестве второго кластера и менее плотную область в центре в качестве третьего кластера. Вместо этого, у кластера 0 и кластера 1 есть несколько точек, которые сильно удалены от всех остальных точек этих кластеров, «тянущихся» к центру.

Кроме того, алгоритм **k-средних** предполагает, что все направления одинаково важны для каждого кластера. Следующий график (рис. 16.6) показывает двумерный набор данных с тремя четко обособленными группами данных. Однако эти группы вытянуты по диагонали. Поскольку алгоритм **k-средних** учитывает лишь расстояние до ближайшего центра кластера, он не может обработать данные такого рода:

```
# генерируем случайным образом данные для кластеризации
X, y = make_blobs(random_state=170, n_samples=600)
```

```
rng = np.random.RandomState(74)
# преобразуем данные так, чтобы они были вытянуты по диагонали
transformation = rng.normal(size=(2, 2))
X = np.dot(X, transformation)

# группируем данные в три кластера
kmeans = KMeans(n_clusters=3)
kmeans.fit(X)
y_pred = kmeans.predict(X)

# строим график принадлежности к кластерам и центров кластеров
plt.scatter(X[:, 0], X[:, 1], c=y_pred, cmap=mglearn.cm3)
plt.scatter(kmeans.cluster_centers_[:, 0],
            kmeans.cluster_centers_[:, 1],
            marker='^', c=[0, 1, 2], s=100, linewidth=2,
            cmap=mglearn.cm3)
plt.xlabel("Признак 0")
plt.ylabel("Признак 1")
plt.show()
```

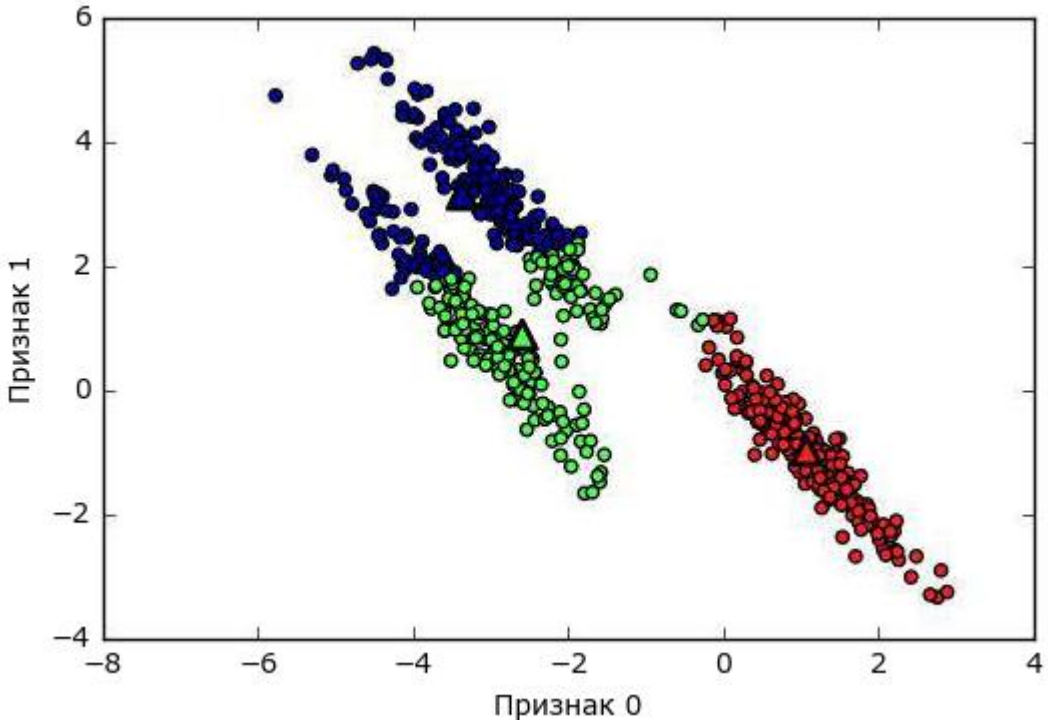

Рис. 16.6 Алгоритм k-средних не позволяет выявить несферические кластеры

Кроме того, алгоритм **k-средних** плохо работает, когда кластеры имеют более сложную форму, как в случае с данными two_moons, с которыми мы столкнулись в главе 2 (см. рис. 16.7):

```
# генерируем синтетические данные two_moons (на этот раз с меньшим количеством шума)
```

```
from sklearn.datasets import make_moons

X, y = make_moons(n_samples=200, noise=0.05, random_state=0)
# группируем данные в два кластера
kmeans = KMeans(n_clusters=2)
kmeans.fit(X)
y_pred = kmeans.predict(X)
# строим график принадлежности к кластерам и центров кластеров
plt.scatter(X[:, 0], X[:, 1], c=y_pred, cmap=mglearn.cm2, s=60)
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1],
            marker='^', c=[mglearn.cm2(0), mglearn.cm2(1)], s=100, linewidth=2)
plt.xlabel("Признак 0")
plt.ylabel("Признак 1")
plt.show()
```

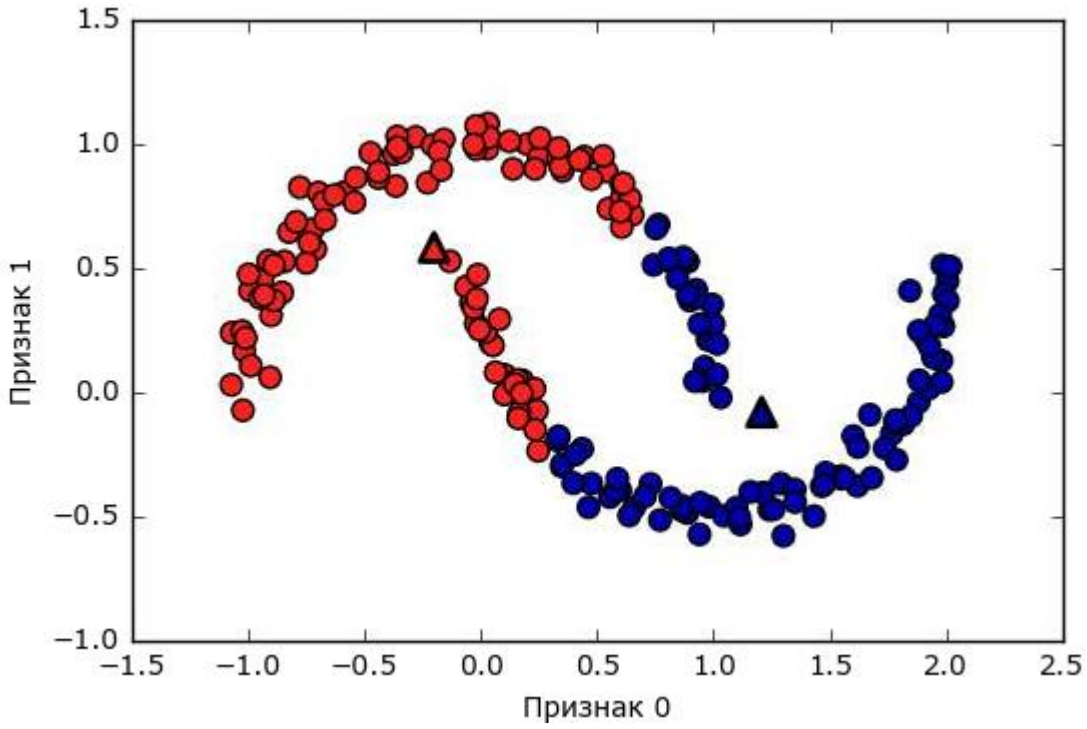

Рис. 16.7 Алгоритм k-средних не позволяет выявить кластеры более сложной формы

В данном случае мы понадеялись на то, что алгоритм кластеризации сможет обнаружить два кластера в форме полумесяцев. Однако определить их с помощью алгоритма **k-средних** не представляется возможным.

Код к лабораторной работе:

```
import mglearn
import sklearn
import matplotlib.pyplot as plt
import numpy as np
from sklearn.model_selection import train_test_split

mglearn.plots.plot_kmeans_algorithm()
plt.show()

mglearn.plots.plot_kmeans_boundaries()
plt.show()

from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans

# генерируем синтетические двумерные данные
X, y = make_blobs(random_state=1)
kmeans = KMeans(n_clusters=3)
kmeans.fit(X)
# строим модель кластеризации

print("Принадлежность к кластерам:\n{}".format(kmeans.labels_))



mglearn.discrete_scatter(X[:, 0], X[:, 1], kmeans.labels_, markers='o')
mglearn.discrete_scatter(kmeans.cluster_centers_[:, 0],
                         kmeans.cluster_centers_[:, 1], [0, 1, 2], markers='^', markeredgewidth=2)
plt.show()

fig, axes = plt.subplots(1, 2, figsize=(10, 5))
#  использование  двух  центров  кластеров:
kmeans = KMeans(n_clusters=2)
kmeans.fit(X)
assignments = kmeans.labels_
mglearn.discrete_scatter(X[:, 0], X[:, 1], assignments, ax=axes[0])
#  использование  пяти  центров  кластеров:
kmeans = KMeans(n_clusters=5)
kmeans.fit(X)
assignments = kmeans.labels_
mglearn.discrete_scatter(X[:, 0], X[:, 1], assignments, ax=axes[1])
plt.show()



X_varied, y_varied = make_blobs(n_samples=200,
                                cluster_std=[1.0, 2.5, 0.5], random_state=170)
y_pred = KMeans(n_clusters=3, random_state=0).fit_predict(X_varied)
mglearn.discrete_scatter(X_varied[:, 0], X_varied[:, 1], y_pred)
```

```
plt.legend(["кластер 0", "кластер 1", "кластер 2"], loc='best')
plt.xlabel("Признак 0")
plt.ylabel("Признак 1")
plt.show()


# генерируем случайным образом данные для кластеризации
X, y = make_blobs(random_state=170, n_samples=600)
rng = np.random.RandomState(74)
# преобразуем данные так, чтобы они были вытянуты по диагонали
transformation = rng.normal(size=(2, 2))
X = np.dot(X, transformation)

# группируем данные в три кластера
kmeans = KMeans(n_clusters=3)
kmeans.fit(X)
y_pred = kmeans.predict(X)

# строим график принадлежности к кластерам и центров кластеров
plt.scatter(X[:, 0], X[:, 1], c=y_pred, cmap=mglearn.cm3)
plt.scatter(kmeans.cluster_centers_[:, 0],
            kmeans.cluster_centers_[:, 1],
            marker='^', c=[0, 1, 2], s=100, linewidth=2,
            cmap=mglearn.cm3)
plt.xlabel("Признак 0")
plt.ylabel("Признак 1")
plt.show()


# генерируем синтетические данные two_moons (на этот раз с меньшим количеством шума)
from sklearn.datasets import make_moons

X, y = make_moons(n_samples=200, noise=0.05, random_state=0)
# группируем данные в два кластера
kmeans = KMeans(n_clusters=2)
kmeans.fit(X)
y_pred = kmeans.predict(X)
# строим график принадлежности к кластерам и центров кластеров
plt.scatter(X[:, 0], X[:, 1], c=y_pred, cmap=mglearn.cm2, s=60)
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1],
            marker='^', c=[mglearn.cm2(0), mglearn.cm2(1)], s=100, linewidth=2)
plt.xlabel("Признак 0")
plt.ylabel("Признак 1")
plt.show()
```