Лаб.3. Заглянем глубже: смеси Гауссовых распределений

Модель кластеризации методом *k*-средних, которую мы обсуждали в предыдущем разделе, проста и относительно легка для понимания, но ее простота приводит к сложностям в применении. В частности, невероятностная природа метода *k*-средних и использование им простого расстояния от центра кластера для определения принадлежности к кластеру приводит к низкой эффективности во многих встречающихся на практике ситуациях. В этом разделе мы рассмотрим смеси Гауссовых распределений, которые можно рассматривать в качестве развития идей метода *k*-средних, но которые могут также стать мощным инструментом для статистических оценок, выходящих за пределы простой кластеризации. Начнем с обычных импортов:

```
In[1]: %matplotlib inline
       import matplotlib.pyplot as plt
       import seaborn as sns; sns.set()
       import numpy as np
```

Причины появления GMM: недостатки метода *k*-средних

Рассмотрим некоторые недостатки метода *k*-средних и задумаемся о том, как можно усовершенствовать кластерную модель. Как мы уже видели в предыдущем разделе, в случае простых, хорошо разделяемых данных метод *k*-средних обеспечивает удовлетворительные результаты кластеризации.

Например, для случая простых «пятен» данных алгоритм *k*-средних позволяет быстро маркировать кластеры достаточно близко к тому, как мы бы маркировали их на глаз (рис. 3.1):

```
In[2]: # Генерируем данные
       from sklearn.datasets.samples_generator import make_blobs
       X, y_true = make_blobs(n_samples=400, centers=4, cluster_std=0.60,
     random_state=0)
       X = X[:, ::-1] # Транспонируем для удобства оси координат
In[3]: # Выводим данные на график с полученными методом k-средних метками
       from sklearn.cluster import KMeans
       kmeans = KMeans(4, random_state=0)
       labels = kmeans.fit(X).predict(X)
       plt.scatter(X[:, 0], X[:, 1], c=labels, s=40, cmap='viridis');
```

Интуитивно мы можем ожидать, что при кластеризации одним точкам метки присваиваются с большей долей достоверности, чем другим. Например, два средних кластера чуть-чуть пересекаются, поэтому нельзя быть уверенными в том, к какому из них отнести точки, находящиеся посередине между ними. К сожалению, в модели *k*-средних отсутствует внутренняя мера вероятности или достоверности отнесения точек к кластерам (хотя можно использовать бутстрэппинг для оценки этой вероятности). Для этого необходимо рассмотреть возможность обобщения модели.

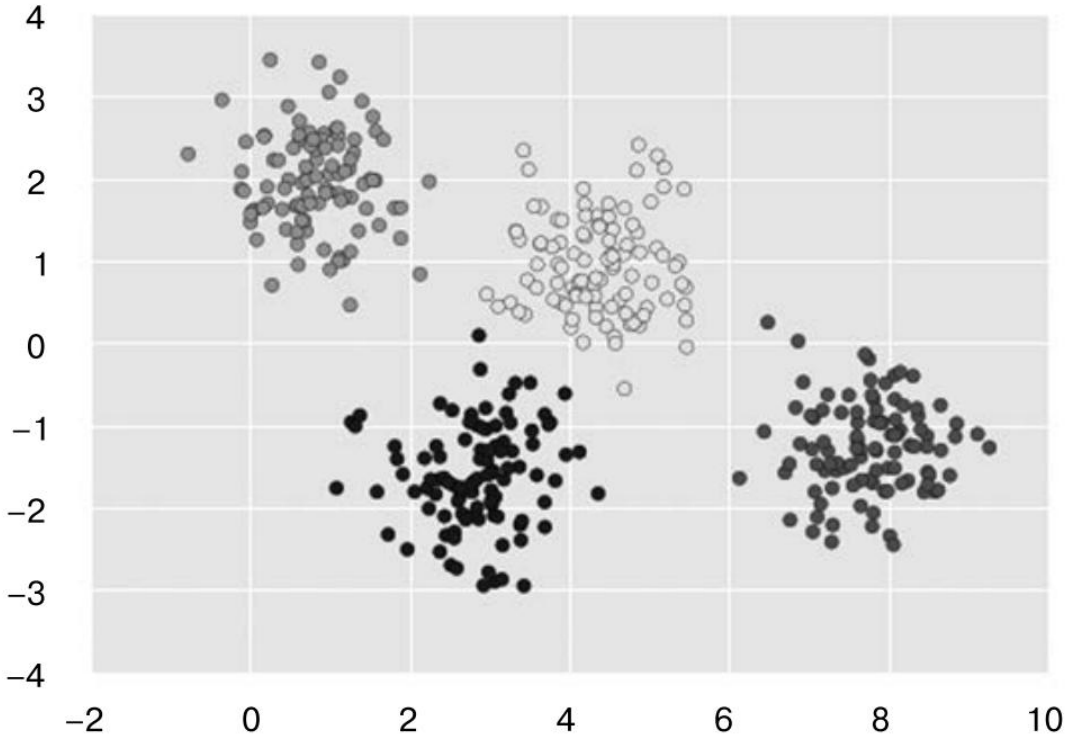

Рис. 3.1. Полученные методом *k*-средних метки для простых данных

Модель *k*-средних можно рассматривать, в частности, как помещающую окружности (или в пространствах большей размерности гиперсферы) с центрами в центрах каждого из кластеров и радиусом, соответствующим расстоянию до наиболее удаленной точки кластера. Этот радиус задает жесткую границу соответствия точки кластеру в обучающей последовательности: все точки, находящиеся снаружи этой окружности, не считаются членами кластера. Визуализируем эту модель кластера с помощью следующей функции (рис. 3.2):

```
In[4]:
from sklearn.cluster import KMeans
from scipy.spatial.distance import cdist

def plot_kmeans(kmeans, X, n_clusters=4, rseed=0, ax=None):
    labels = kmeans.fit_predict(X)
    # Выводим на рисунок входные данные
    ax = ax or plt.gca()
    ax.axis('equal')
    ax.scatter(X[:, 0], X[:, 1], c=labels, s=40, cmap='viridis', zorder=2)
    # Выводим на рисунок представление модели k-средних
    centers = kmeans.cluster_centers_
    radii = [cdist(X[labels == i], [center]).max() for i, center in enumerate(centers)]
    for c, r in zip(centers, radii):
        ax.add_patch(plt.Circle(c, r, fc='#CCCCCC', lw=3, alpha=0.5, zorder=1))
In[5]: kmeans = KMeans(n_clusters=4, random_state=0)
       plot_kmeans(kmeans, X)
```

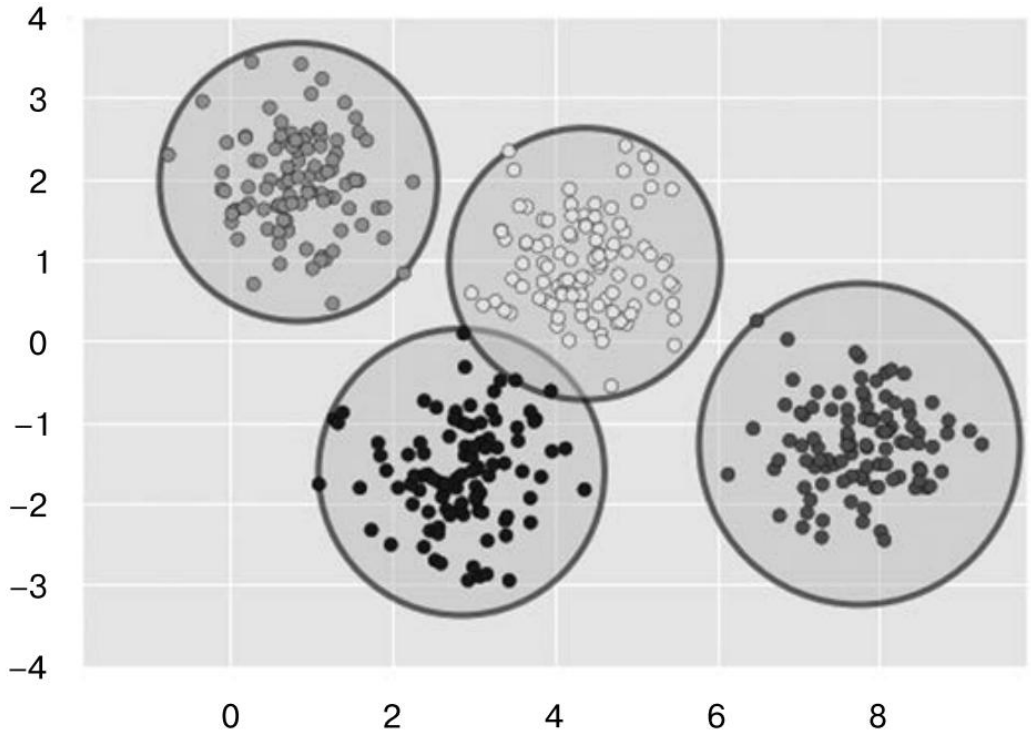

Рис. 3.2. Круглые кластеры, подразумеваемые моделью *k*-средних

Немаловажный нюанс, касающийся метода *k*-средних, состоит в том, что эти модели кластеров обязательно должны иметь форму окружностей: метод *k*-средних не умеет работать с овальными или эллипсовидными кластерами. Так, например, если несколько преобразовать те же данные, присвоенные метки окажутся перепутаны (рис. 3.3).

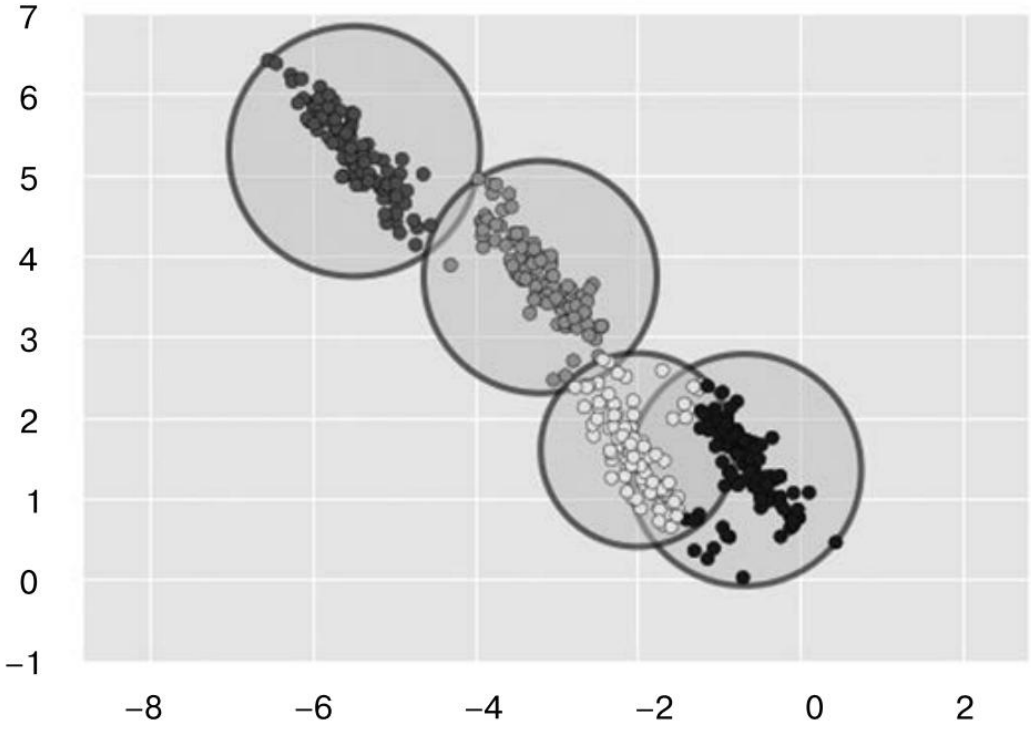

Рис. 3.3. Неудовлетворительная работа метода *k*-средних в случае кластеров некруглой формы

```
In[6]: rng = np.random.RandomState(13)
       X_stretched = np.dot(X, rng.randn(2, 2))
       kmeans = KMeans(n_clusters=4, random_state=0)
       plot_kmeans(kmeans, X_stretched)
```

Визуально заметно, что форма этих преобразованных кластеров некруглая, а значит, круглые кластеры плохо подойдут для их описания. Тем не менее метод *k*-средних недостаточно гибок для учета этого нюанса и пытается втиснуть данные в четыре круглых кластера. Это приводит к перепутанным меткам кластеров в местах перекрытия получившихся окружностей, см. нижнюю правую часть графика.

Можно было бы попытаться решить эту проблему путем предварительной обработки данных с помощью PCA (метод главных компонент, которпый будет рассмотрен в одной из следующих работ), но на практике нет никаких гарантий, что подобная глобальная операция позволит разместить по окружностям отдельные точки данных.

Отсутствие гибкости в вопросе формы кластеров и отсутствие вероятностного присвоения меток кластеров — два недостатка метода *k*-средних, означающих, что для многих наборов данных (особенно низкоразмерных) он будет работать не столь хорошо, как хотелось бы.

Можно попытаться избавиться от этих недостатков путем обобщения модели *k*-средних. Например, можно оценивать степень достоверности присвоения меток кластеров, сравнивая расстояния от каждой точки до всех центров кластеров, а не сосредотачивая внимание лишь на ближайшем. Можно также разрешить эллипсовидную форму границ кластеров, а не только круглую, чтобы учесть кластеры некруглой формы. Оказывается, что это базовые составляющие другой разновидности модели кластеризации — смеси Гауссовых распределений.

Обобщение EM-модели: смеси Гауссовых распределений

Смесь Гауссовых распределений (gaussian mixture model, GMM) нацелена на поиск многомерных Гауссовых распределений вероятностей, моделирующих наилучшим возможным образом любой исходный набор данных.

В простейшем случае смеси Гауссовых распределений можно использовать для поиска кластеров аналогично методу *k*-средних (рис. 3.4):

```
In[7]: from sklearn.mixture import GMM
       gmm = GMM(n_components=4).fit(X)
       labels = gmm.predict(X)
       plt.scatter(X[:, 0], X[:, 1], c=labels, s=40, cmap='viridis');
```

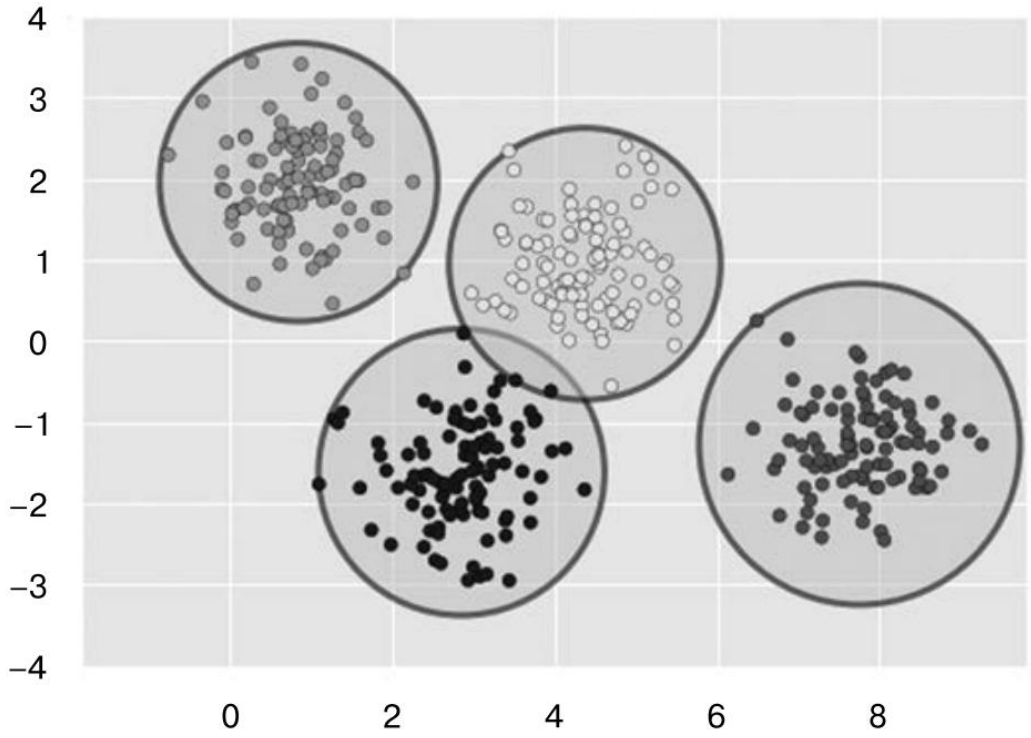

Рис. 3.4. Метки смеси Гауссовых распределений для наших данных

Но в силу того, что GMM содержит «под капотом» вероятностную модель, с ее помощью можно также присваивать метки кластеров на вероятностной основе — в библиотеке Scikit-Learn это можно сделать методом predict_proba. Он возвращает матрицу размера [n_samples, n_clusters], содержащую оценки вероятностей принадлежности точки к конкретному кластеру:

```
In[8]: probs = gmm.predict_proba(X)
       print(probs[:5].round(3))

[[ 0. 0.     0.475    0.525 ]
 [ 0.1.      0.       0.    ]
 [ 0.1.      0.       0.    ]
 [ 0.0.      0.       1.    ]
 [ 0.1.      0.       0.    ]]
```

Для визуализации этой вероятности можно, например, сделать размеры точек пропорциональными степени достоверности их предсказания. Глядя на рис. 3.5, можно увидеть, что как раз точки на границах между кластерами отражают эту неопределенность отнесения точек к кластерам:

```
In[9]: size = 50 * probs.max(1) ** 2 # Возведение в квадрат усиливает влияние
                                     # различий
       plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis', s=size);
```

«Под капотом» смесь Гауссовых распределений очень напоминает метод *k*-средних: она использует подход с максимизацией математического ожидания, который с качественной точки зрения делает следующее.

1. Выбирает первоначальные гипотезы для расположения и формы кластеров.
2. Повторяет до достижения сходимости:
   - E-шаг — для каждой точки находит веса, кодирующие вероятность ее принадлежности к каждому кластеру;

- М-шаг — для каждого кластера корректирует его расположение, нормализацию и форму на основе информации обо *всех* точках данных с учетом весов.

В результате каждый кластер оказывается связан не со сферой с четкой границей, а с гладкой Гауссовой моделью. Аналогично подходу максимизации математического ожидания из метода *k*-средних этот алгоритм иногда может промахиваться мимо наилучшего из возможных решений, поэтому на практике применяют несколько случайных наборов начальных значений.

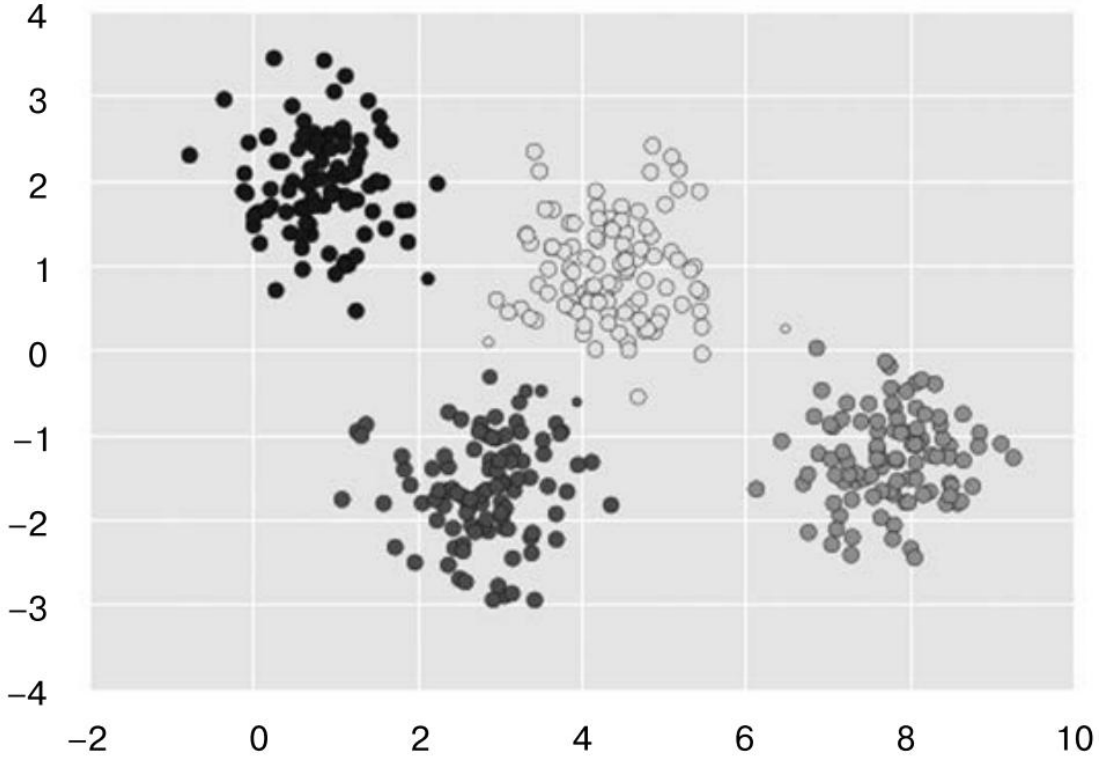

Рис. 3.5. Вероятностные метки GMM

Создадим функцию для упрощения визуализации расположений и форм кластеров метода GMM путем рисования эллипсов на основе получаемой на выходе gmm информации:

```
In[10]:
from matplotlib.patches import Ellipse
def draw_ellipse(position, covariance, ax=None, **kwargs):
    """Рисует эллипс с заданными расположением и ковариацией"""
    ax = ax or plt.gca()

    # Преобразуем ковариацию к главным осям координат
    if covariance.shape == (2, 2):
        U, s, Vt = np.linalg.svd(covariance)
        angle = np.degrees(np.arctan2(U[1, 0], U[0, 0]))
        width, height = 2 * np.sqrt(s)
    else:
        angle = 0
        width, height = 2 * np.sqrt(covariance)

        # Рисуем эллипс
```

```
            for nsig in range(1, 4):
                ax.add_patch(Ellipse(position, nsig * width, nsig * height, angle,
                             **kwargs))

def plot_gmm(gmm, X, label=True, ax=None):
    ax = ax or plt.gca()
    labels = gmm.fit(X).predict(X)
    if label:
        ax.scatter(X[:, 0], X[:, 1], c=labels, s=40, cmap='viridis', zorder=2)
    else:
        ax.scatter(X[:, 0], X[:, 1], s=40, zorder=2)
    ax.axis('equal')
    w_factor = 0.2 / gmm.weights_.max()
    for pos, covar, w in zip(gmm.means_, gmm.covars_, gmm.weights_):
        draw_ellipse(pos, covar, alpha=w * w_factor)
```

После этого можем посмотреть, какие результаты выдает четырехкомпонентный метод GMM на наших данных (рис. 3.6):

```
In[11]: gmm = GMM(n_components=4, random_state=42)
        plot_gmm(gmm, X)
```

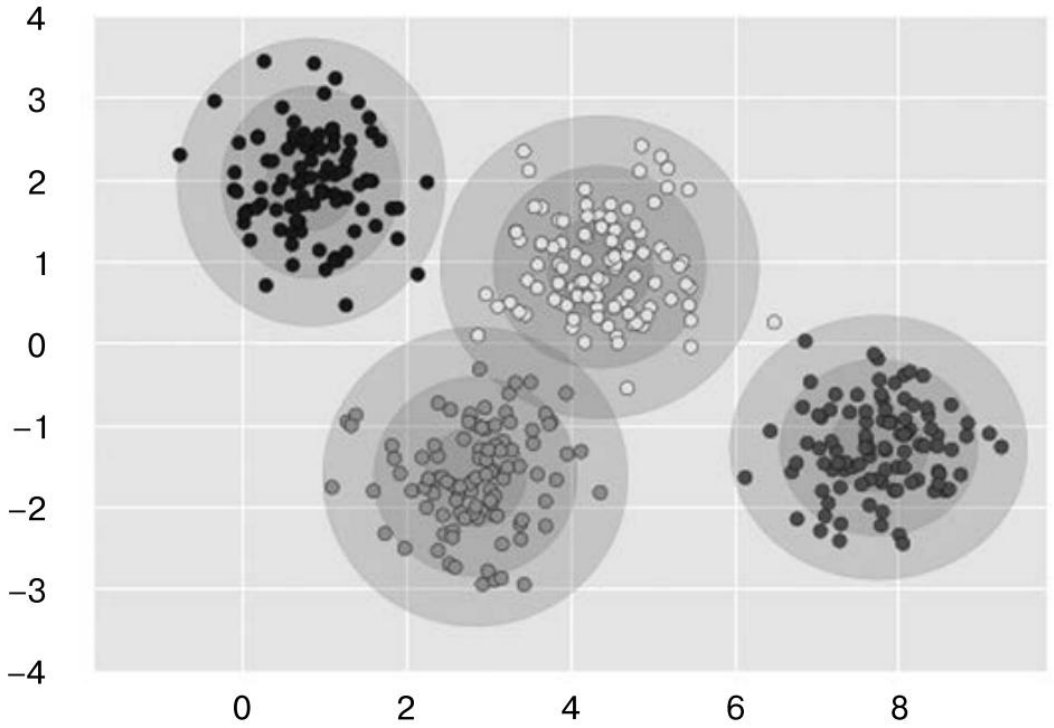

Рис. 3.6. Четырехкомпонентный метод GMM в случае круглых кластеров

Аналогично можно воспользоваться подходом GMM для «растянутого» набора данных. С учетом полной ковариации модель будет подходить даже для очень продолговатых, вытянутых в длину кластеров (рис. 3.7):

```
In[12]: gmm = GMM(n_components=4, covariance_type='full', random_state=42)
        plot_gmm(gmm, X_stretched)
```

Из этого ясно, что метод GMM решает две известные нам основные практические проблемы метода *k*-средних.

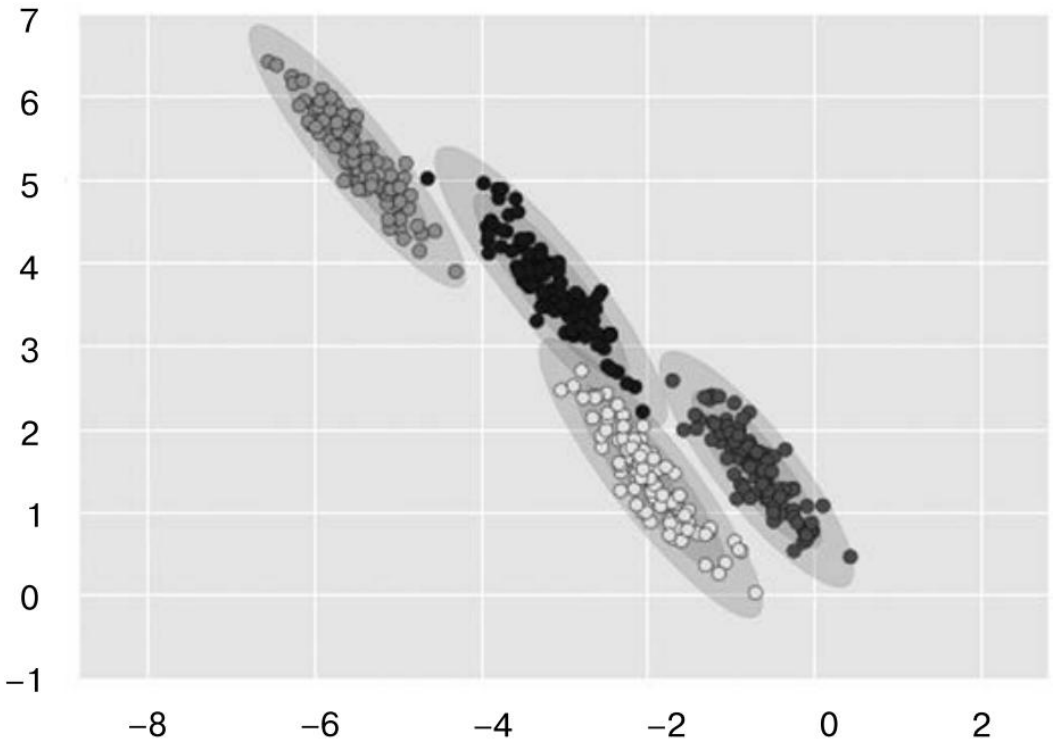

Рис. 3.7. Четырехкомпонентный метод GMM в случае некруглых кластеров

Выбор типа ковариации

Если вы внимательно посмотрите на предыдущие фрагменты кода, то увидите, что в каждом из них были заданы различные значения параметра covariance_type. Этот гиперпараметр управляет степенями свободы форм кластеров. Очень важно для любой задачи задавать его значения аккуратно. Значение его по умолчанию — covariance_type="diag", означающее возможность независимого задания размеров кластера по всем измерениям с выравниванием полученного эллипса по осям координат.

Несколько более простая и быстро работающая модель — covariance_type="spherical", ограничивающая форму кластера таким образом, что все измерения равнозначны между собой. Получающаяся в этом случае кластеризация будет аналогична методу *k*-средних, хотя и не полностью идентична.

Вариант с covariance_type="full" представляет собой более сложную и требующую больших вычислительных затрат модель (особенно при росте числа измерений), в которой любой из кластеров может быть эллипсом с произвольной ориентацией.

Графическое представление этих трех вариантов для одного кластера приведено на рис. 3.8.

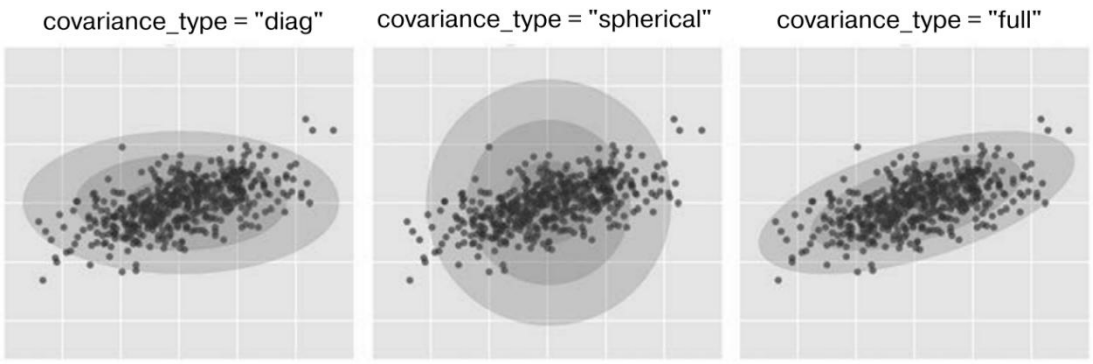

Рис. 3.8. Визуализация типов ковариации метода GMM

GMM как метод оценки плотности распределения

Хотя GMM часто относят к алгоритмам кластеризации, по существу, это алгоритм, предназначенный для оценки плотности распределения. Таким образом, аппроксимация каких-либо данных методом GMM формально является не моделью кластеризации, а порождающей вероятностной моделью, описывающей распределение данных.

В качестве примера изучим данные, сгенерированные с помощью функции make_moons библиотеки Scikit-Learn (показанные на рис. 3.9), которые мы уже рассматривали в лабораторной работе «Заглянем глубже: кластеризация методом *k*-средних».

```
In[13]: from sklearn.datasets import make_moons
        Xmoon, ymoon = make_moons(200, noise=.05, random_state=0)
        plt.scatter(Xmoon[:, 0], Xmoon[:, 1]);
```

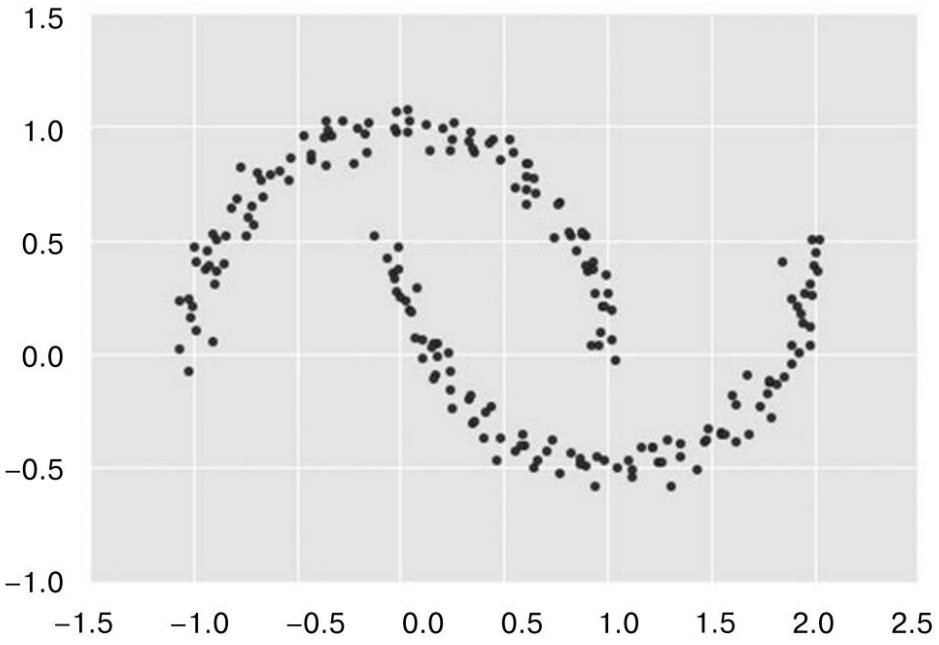

Рис. 3.9. Использование метода GMM для кластеров с нелинейными границами

Если попытаться использовать для этих данных двухкомпонентный GMM, рассматриваемый как модель кластеризации, то практическая пригодность результатов окажется сомнительной (рис. 3.10):

```
In[14]: gmm2 = GMM(n_components=2, covariance_type='full', random_state=0)
```

```
plot_gmm(gmm2, Xmoon)
```

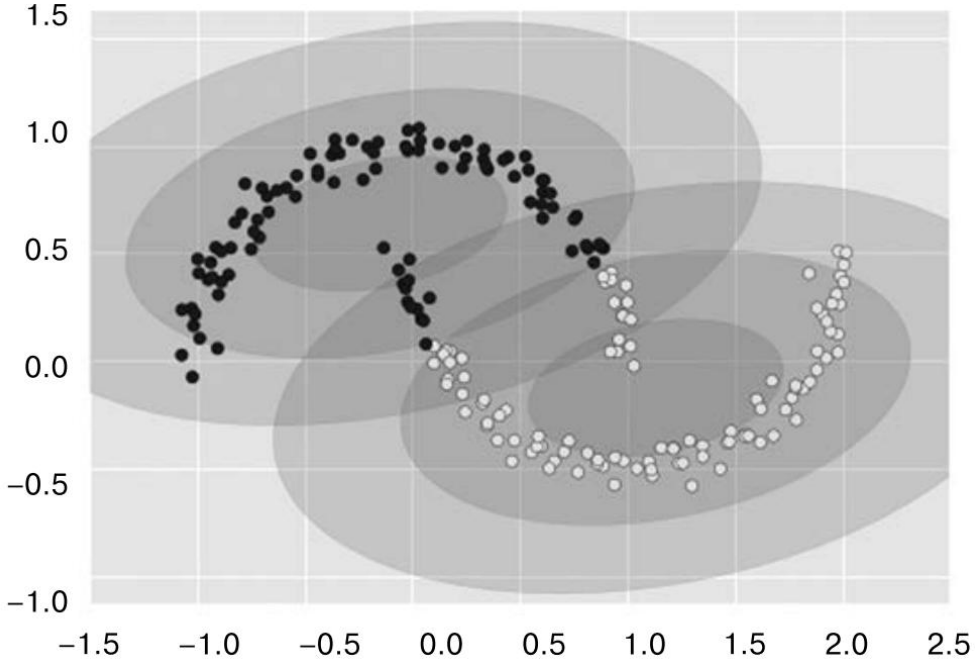

Рис. 3.10. Двухкомпонентная GMM-аппроксимация в случае нелинейных кластеров

Но если взять намного больше компонент и проигнорировать метки кластеров, мы получим намного более подходящую для исходных данных аппроксимацию (рис. 3.11):

```
In[15]: gmm16 = GMM(n_components=16, covariance_type='full', random_state=0)
        plot_gmm(gmm16, Xmoon, label=False)
```

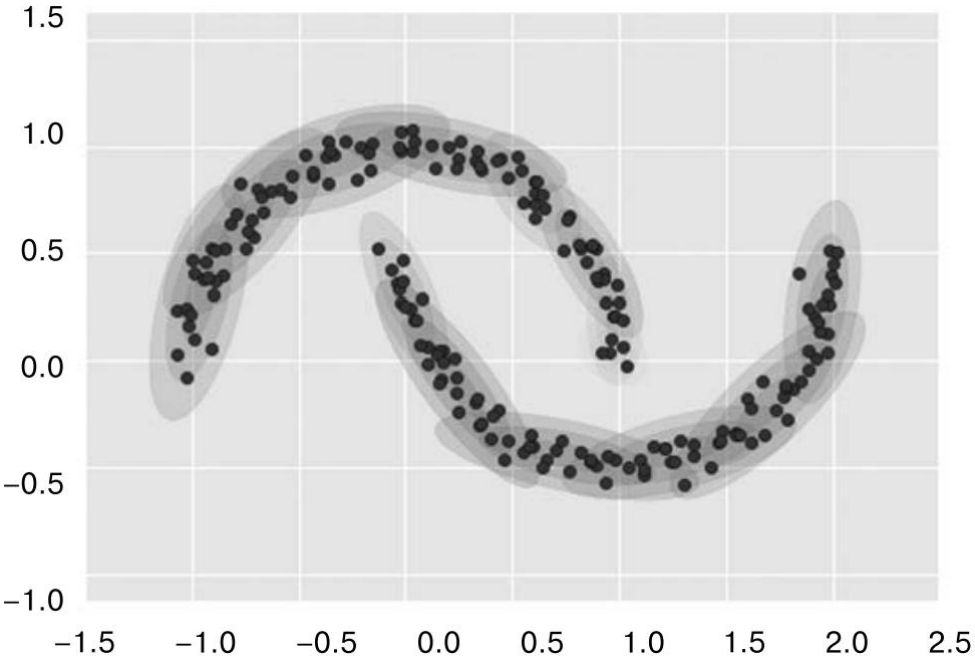

Рис. 3.11. Использование большого количества кластеров GMM для моделирования распределения точек

В данном случае смесь 16 нормальных распределений служит не для поиска отдельных кластеров данных, а для моделирования общего распределения входных данных. Это порождающая модель распределения, то есть GMM предоставляет нам способ генерации новых случайных данных, распределенных аналогично исходным. Например, вот 400 новых точек, полученных из этой аппроксимации наших исходных данных 16-компонентным алгоритмом GMM (рис. 3.12):

```
In[16]: Xnew = gmm16.sample(400, random_state=42)
        plt.scatter(Xnew[:, 0], Xnew[:, 1]);
```

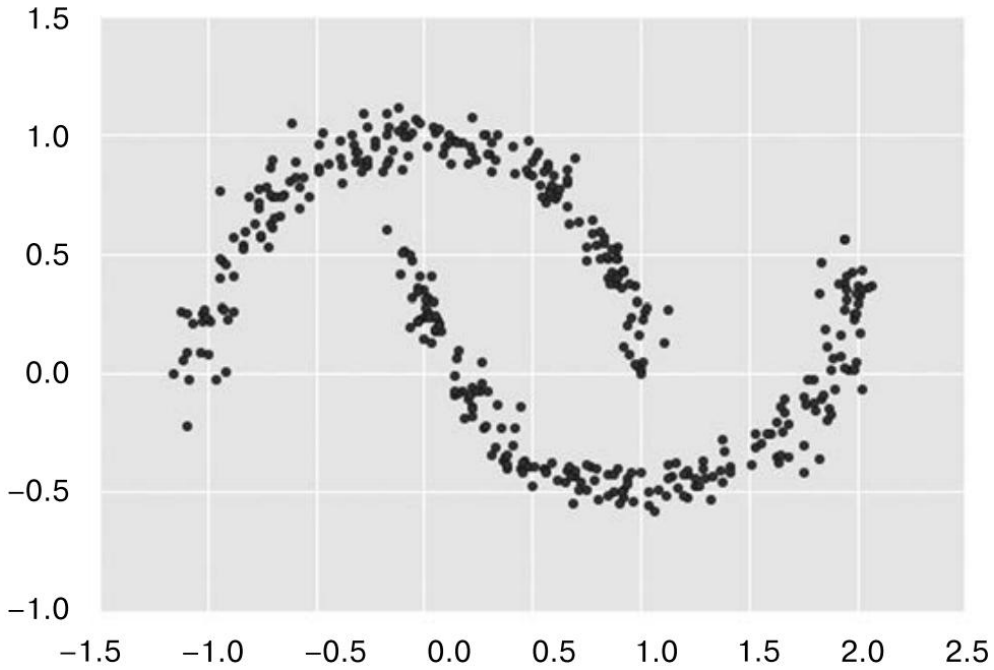

Рис. 3.12. Новые данные, полученные из 16-компонентного GMM

Метод GMM — удобное гибкое средство моделирования произвольного многомерного распределения данных.

Сколько компонент необходимо?

Благодаря тому что GMM — порождающая модель, у нас появляется естественная возможность определения оптимального количества компонент для заданного набора данных. Порождающая модель, по существу, представляет собой распределение вероятности для набора данных, поэтому можно легко вычислить функцию правдоподобия (likelihood function) для лежащих в ее основе данных, используя перекрестную проверку во избежание переобучения. Другой способ введения поправки на переобучение — подстройка функции правдоподобия модели с помощью некоторого аналитического критерия, например информационного критерия Акаике (Akaike information criterion, AIC, см.: https://ru.wikipedia.org/wiki/Информационный_критерий_Акаике) или байесовского информационного критерия (bayesian information criterion, BIC, см.: https://ru.wikipedia.org/wiki/Информационный_критерий). Оцениватель GMM библиотеки Scikit-Learn включает встроенные методы для вычисления этих критериев, что сильно упрощает указанный подход.

Посмотрим на критерии AIC и BIC как функции от количества компонент GMM для нашего набора данных moon (рис. 3.13):

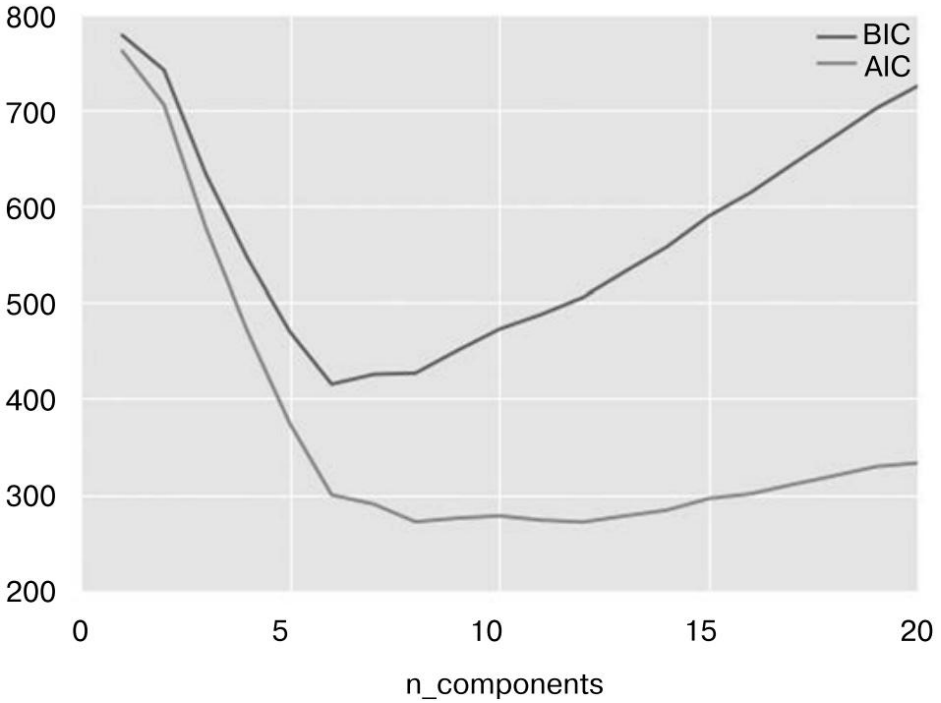

Рис. 3.13. Визуализация AIC и BIC с целью выбора количества компонент GMM

```
In[17]: n_components = np.arange(1, 21)
        models = [GMM(n, covariance_type='full', random_state=0).fit(Xmoon)
        for n in n_components]
            plt.plot(n_components, [m.bic(Xmoon) for m in models], label='BIC')
            plt.plot(n_components, [m.aic(Xmoon) for m in models], label='AIC')
            plt.legend(loc='best')
            plt.xlabel('n_components');
```

Оптимальное количество кластеров — то, которое минимизирует AIC или BIC, в зависимости от требуемой аппроксимации. Согласно AIC, наших 16 компонент, вероятно, слишком много, лучше взять 8–12. Как это обычно бывает в подобных задачах, критерий BIC говорит в пользу более простой модели.

Обратите внимание на важный момент: подобный метод выбора числа компонент представляет собой меру успешности работы GMM как оценивателя плотности распределения, а не как алгоритма кластеризации. Лучше всего рассматривать GMM в основном как оцениватель плотности и использовать его для кластеризации только заведомо простых наборов данных.

Пример: использование метода GMM для генерации новых данных

Мы увидели простой пример применения метода GMM в качестве порождающей модели данных с целью создания новых выборок на основе соответствующего исходным данным распределения. В этом разделе мы продолжим воплощение этой идеи и сгенерируем новые рукописные цифры на основе корпуса

стандартных цифр, который мы использовали ранее. Для начала загрузим набор данных по цифрам с помощью инструментов библиотеки Scikit-Learn:

```
In[18]: from sklearn.datasets import load_digits
        digits = load_digits()
        digits.data.shape
Out[18]: (1797, 64)
```

Далее выведем на рисунок первые 100 из них, чтобы вспомнить, с чем мы имеем дело (рис. 3.14):

```
In[19]: def plot_digits(data):
            fig, ax = plt.subplots(10, 10, figsize=(8, 8),
            subplot_kw=dict(xticks=[], yticks=[]))
            fig.subplots_adjust(hspace=0.05, wspace=0.05)
            for i, axi in enumerate(ax.flat):
                im = axi.imshow(data[i].reshape(8, 8), cmap='binary')
            im.set_clim(0, 16)
        plot_digits(digits.data)
```

Наш набор данных состоит почти из 1800 цифр в 64 измерениях. Построим на их основе GMM, чтобы сгенерировать еще. У смесей Гауссовых распределений могут быть проблемы со сходимостью в пространстве столь высокой размерности, поэтому начнем с применения обратимого алгоритма для понижения размерности данных. Воспользуемся для этой цели простым алгоритмом PCA (методом главных компонент) с сохранением 99% дисперсии в проекции данных:

```
In[20]: from sklearn.decomposition import PCA
        pca = PCA(0.99, whiten=True)
        data = pca.fit_transform(digits.data)
        data.shape
Out[20]: (1797, 41)
```

Результат оказался 41-мерным, то есть размерность была снижена почти на 1/3 практически без потерь информации. Воспользуемся для этих спроецированных данных критерием AIC для определения необходимого количества компонент GMM (рис. 3.15):

```
In[21]: n_components = np.arange(50, 210, 10)
        models = [GMM(n, covariance_type='full', random_state=0)
                              for n in n_components]
        aics = [model.fit(data).aic(data) for model in models]
        plt.plot(n_components, aics);
```

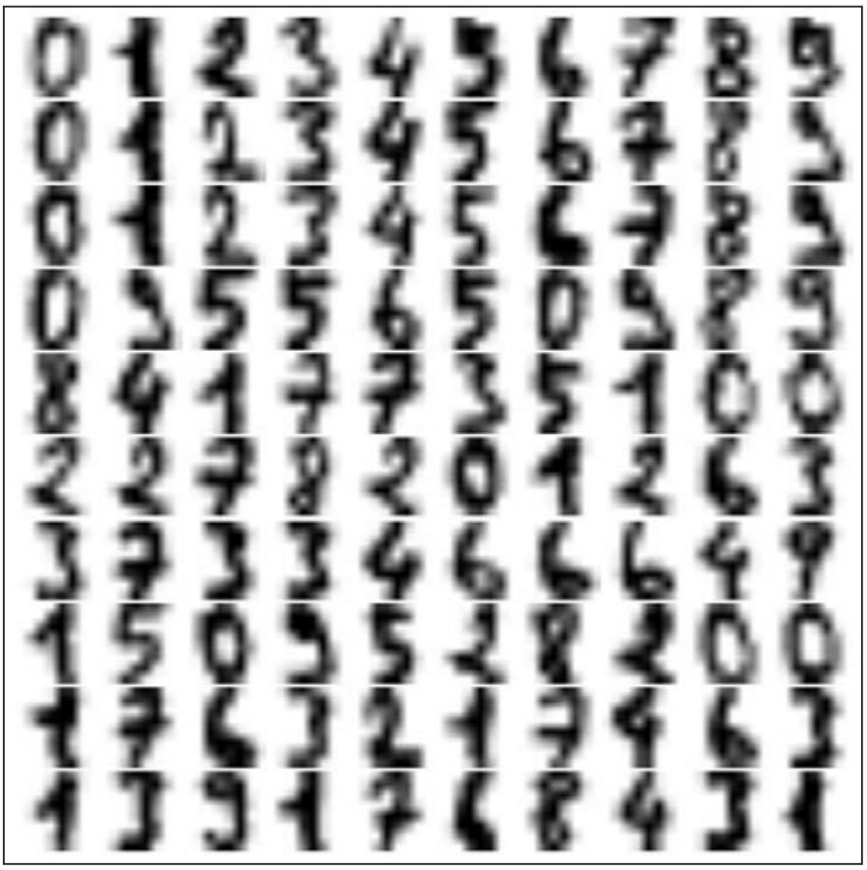

Рис. 3.14. Исходные рукописные цифры

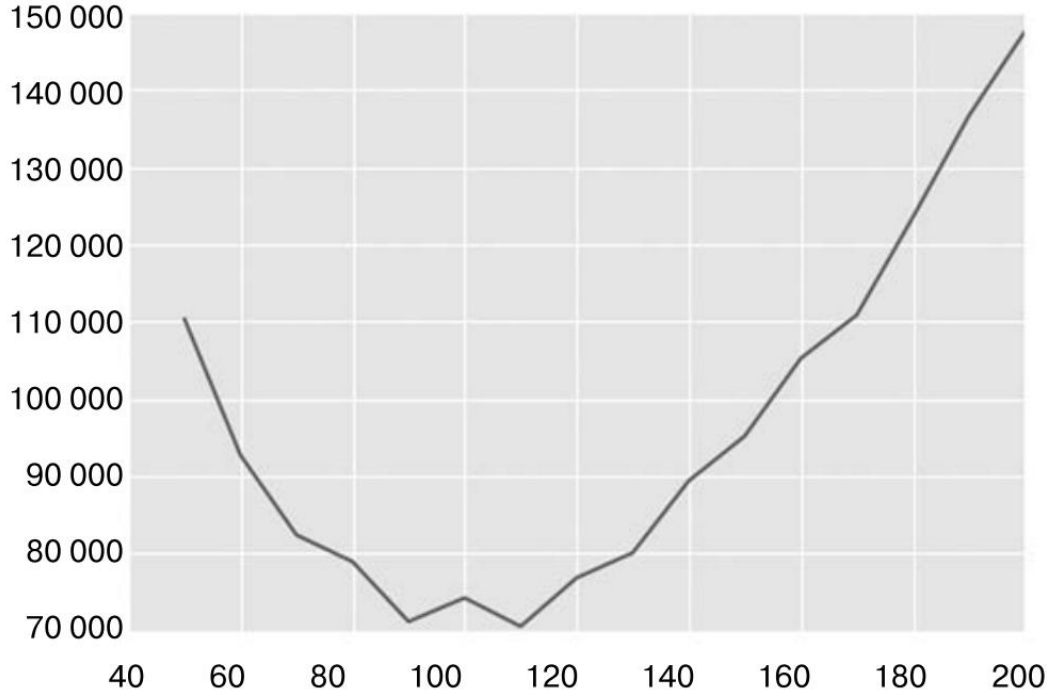

Рис. 3.15. Кривая AIC для выбора подходящего количества компонент GMM

Похоже, что AIC минимизируют примерно 110 компонент; этой моделью мы и воспользуемся. Обучим этот алгоритм на наших данных и убедимся, что он сошелся:

```
In[22]: gmm = GMM(110, covariance_type='full', random_state=0)
        gmm.fit(data)
        print(gmm.converged_)
True
```

Теперь можно сгенерировать 100 новых точек в этом 41-мерном пространстве, используя GMM как порождающую модель:

```
In[23]: data_new = gmm.sample(100, random_state=0)
        data_new.shape
Out[23]: (100, 41)
```

Наконец, можно воспользоваться обратным преобразованием объекта PCA для формирования новых цифр (рис. 3.16):

```
In[24]: digits_new = pca.inverse_transform(data_new)
        plot_digits(digits_new)
```

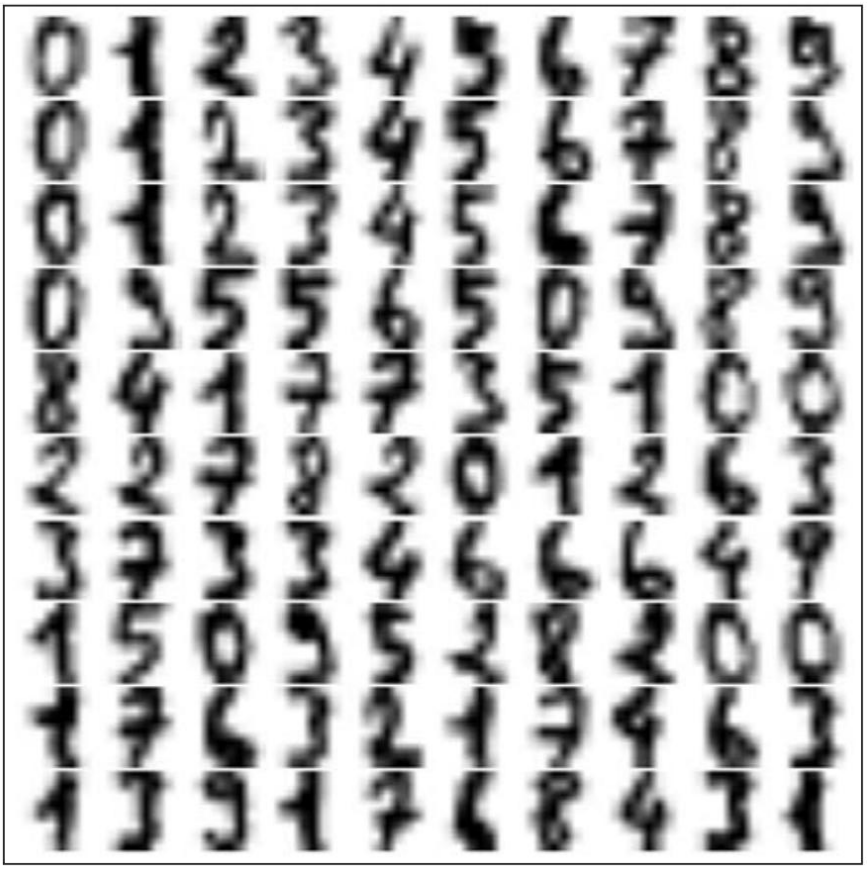

Рис. 3.16. «Новые» цифры, полученные случайным образом из модели оценивателя GMM

Результаты по большей части выглядят как вполне правдоподобные цифры из набора данных! Резюмируем сделанное: мы смоделировали распределение для

заданной выборки рукописных цифр таким образом, что смогли сгенерировать совершенно новые выборки цифр на основе этих данных: это рукописные цифры, не встречающиеся в исходном наборе данных, но отражающие общие признаки входных данных, смоделированные моделью смеси распределений. Подобная порождающая модель цифр может оказаться очень удобной в качестве компонента байесовского порождающего классификатора.