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

## Motivating GMM: Weaknesses of k-Means

Let's take a look at some of the weaknesses of *k*-means and think about how we might improve the cluster model. As we saw in the previous section, given simple, well-separated data, *k*-means finds suitable clustering results.

For example, if we have simple blobs of data, the *k*-means algorithm can quickly label those clusters in a way that closely matches what we might do by eye (Figure 5-124):

```
In[2]: # Generate some data
       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] # flip axes for better plotting
```

```
In[3]: # Plot the data with k-means labels
       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');
```

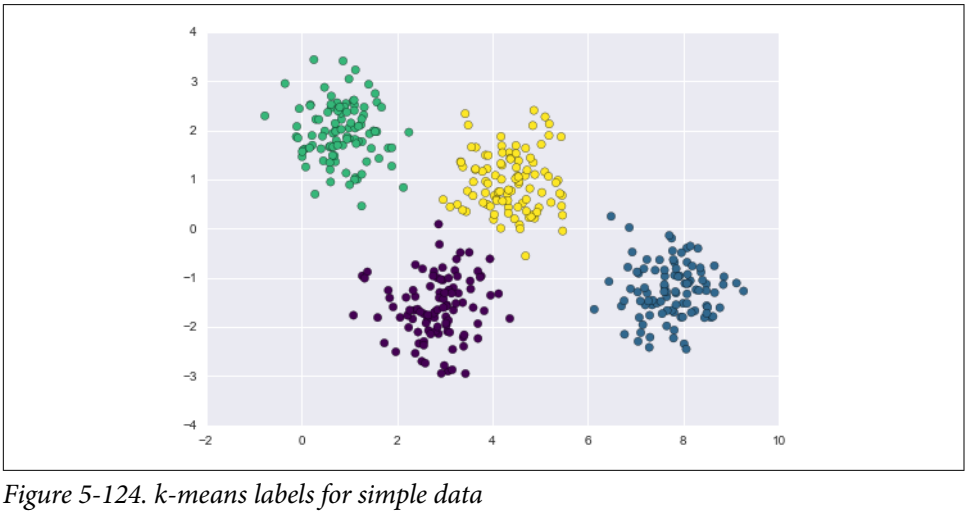


*Figure 5-124. k-means labels for simple data*

From an intuitive standpoint, we might expect that the clustering assignment for some points is more certain than others; for example, there appears to be a very slight overlap between the two middle clusters, such that we might not have complete confidence in the cluster assignment of points between them. Unfortunately, the *k*-means model has no intrinsic measure of probability or uncertainty of cluster assignments

(although it may be possible to use a bootstrap approach to estimate this uncertainty). For this, we must think about generalizing the model.

One way to think about the *k*-means model is that it places a circle (or, in higher dimensions, a hyper-sphere) at the center of each cluster, with a radius defined by the most distant point in the cluster. This radius acts as a hard cutoff for cluster assignment within the training set: any point outside this circle is not considered a member of the cluster. We can visualize this cluster model with the following function (Figure 5-125):

```
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)

    # plot the input data
    ax = ax or plt.gca()
    ax.axis('equal')
    ax.scatter(X[:, 0], X[:, 1], c=labels, s=40, cmap='viridis', zorder=2)

    # plot the representation of the k-means model
    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)
```

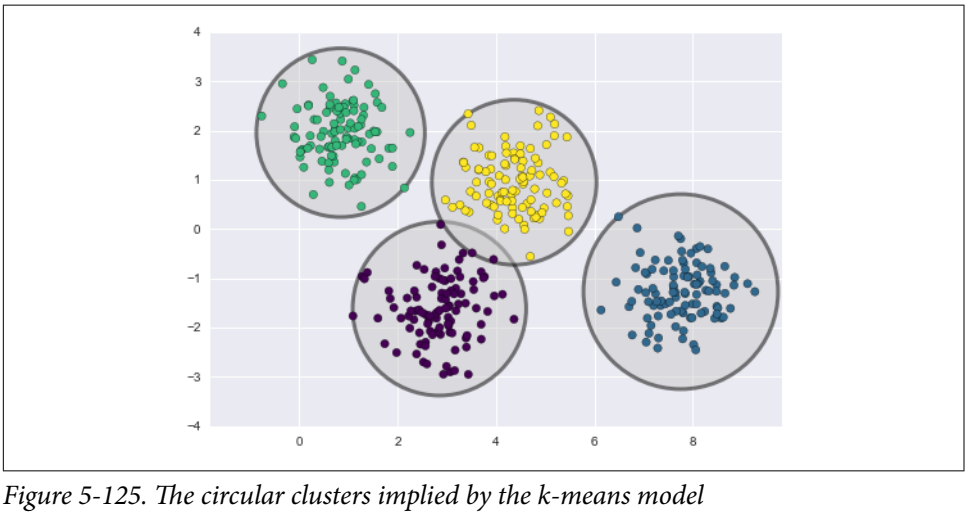


*Figure 5-125. The circular clusters implied by the k-means model*

An important observation for *k*-means is that these cluster models *must be circular*: *k*-means has no built-in way of accounting for oblong or elliptical clusters. So, for example, if we take the same data and transform it, the cluster assignments end up becoming muddled (Figure 5-126):

```
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)
```

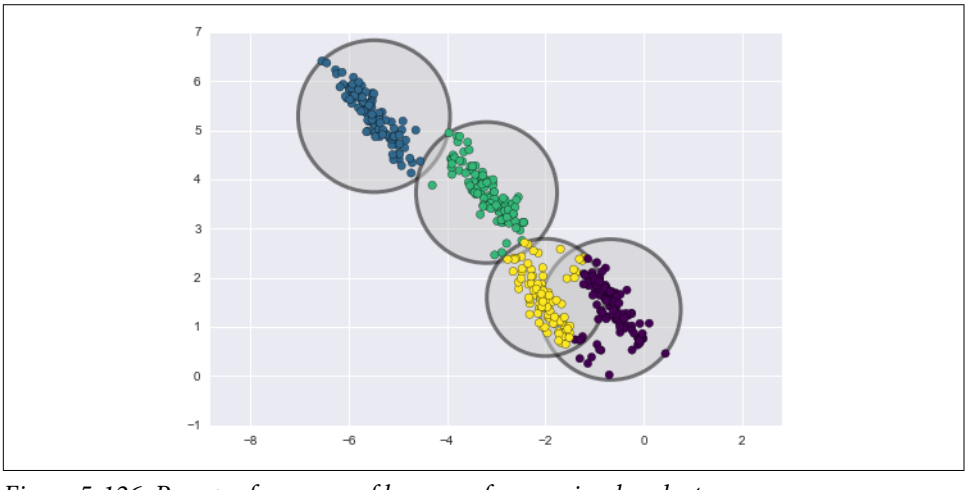


*Figure 5-126. Poor performance of k-means for noncircular clusters*

By eye, we recognize that these transformed clusters are noncircular, and thus circular clusters would be a poor fit. Nevertheless, *k*-means is not flexible enough to account for this, and tries to force-fit the data into four circular clusters. This results in a mixing of cluster assignments where the resulting circles overlap: see especially the bottom right of this plot. One might imagine addressing this particular situation by preprocessing the data with PCA (see "In Depth: Principal Component Analysis" on page 433), but in practice there is no guarantee that such a global operation will circularize the individual data.

These two disadvantages of *k*-means—its lack of flexibility in cluster shape and lack of probabilistic cluster assignment—mean that for many datasets (especially low-dimensional datasets) it may not perform as well as you might hope.

You might imagine addressing these weaknesses by generalizing the *k*-means model: for example, you could measure uncertainty in cluster assignment by comparing the distances of each point to *all* cluster centers, rather than focusing on just the closest. You might also imagine allowing the cluster boundaries to be ellipses rather than cir-

cles, so as to account for noncircular clusters. It turns out these are two essential components of a different type of clustering model, Gaussian mixture models.

## Generalizing E–M: Gaussian Mixture Models

A Gaussian mixture model (GMM) attempts to find a mixture of multidimensional Gaussian probability distributions that best model any input dataset. In the simplest case, GMMs can be used for finding clusters in the same manner as *k*-means (Figure 5-127):

```
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');
```

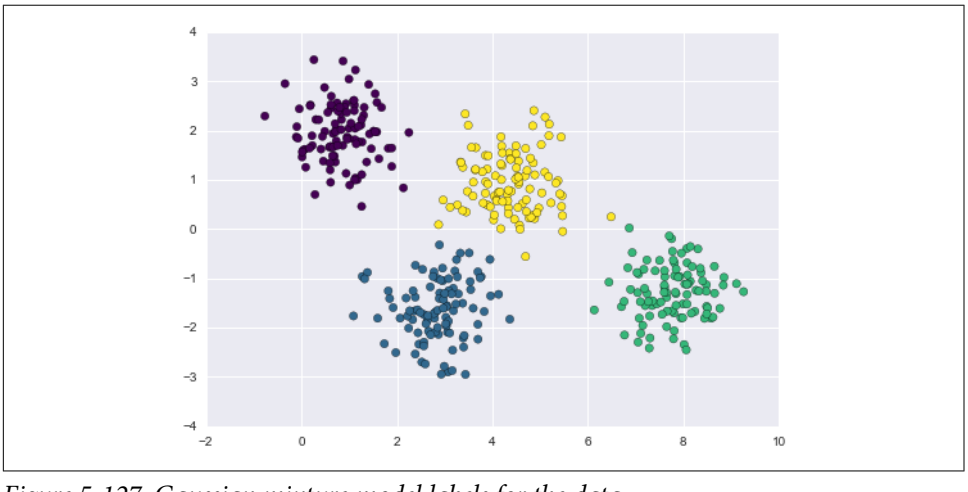


*Figure 5-127. Gaussian mixture model labels for the data*

But because GMM contains a probabilistic model under the hood, it is also possible to find probabilistic cluster assignments—in Scikit-Learn we do this using the `predict_proba` method. This returns a matrix of size `[n_samples, n_clusters]` that measures the probability that any point belongs to the given cluster:

```
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.   ]]
```

We can visualize this uncertainty by, for example, making the size of each point proportional to the certainty of its prediction; looking at Figure 5-128, we can see that it