Considerations for Selecting *K*

There's really no way of telling the number of clusters in a dataset from the onset. The best way of selecting *k* is to try out different values of *K* to see what works best in creating distinct clusters.

Another strategy, which is widely employed in practice, is to compute the average distance of the points in the cluster to the cluster centroid for all clusters. This estimate is plotted on a graph as we progressively increase the value of *K*. We observe that as *K* increases, the distance of points from the centroid of its cluster gradually reduces, and the generated curve resembles the elbow of an arm. From practice, we choose the value of *K* just after the elbow as the best *K* value for that dataset. This method is called the elbow method for selecting *K* as is illustrated in Figure 25-3.

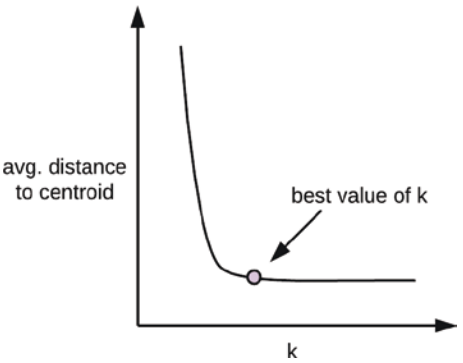

Figure 25-3. *The elbow method for choosing the best value of k*

Considerations for Assigning the Initial *K* Points

The points that determine the initial value of *K* are important in finding a good set of clusters. By selecting the point for *K* at random, two or more points may reside in the same cluster, and this will invariably lead to sub-par results. To mitigate this from occurring, we can employ more sophisticated approaches to selecting the value of *K*. A common strategy is to randomly select the first *K* point and then select the next point as the point that is farthest from the first chosen point. This strategy is repeated until all *K* points have been selected. Another approach is to run hierarchical clustering on a sub-sample of the dataset (this is because hierarchical clustering is a computationally expensive algorithm) and use the number of clusters after cutting off the dendrogram as the value of *K*.

K-Means Clustering with Scikit-learn

This example implements K-means clustering with Scikit-learn. Since this is an unsupervised learning use case, we use just the features of the Iris dataset to cluster the observations into labels.

```
# import packages
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn import datasets
from sklearn.model_selection import train_test_split

# load dataset
data = datasets.load_iris()

# get the dataset features
X = data.data

# create the model. Since we know that the Iris dataset has 3 classes, we
set n_clusters = 3
kmeans = KMeans(n_clusters=3, random_state=0)

# fit the model on the training set
kmeans.fit(X)
'Output':
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
    n_clusters=3, n_init=10, n_jobs=1, precompute_distances='auto',
    random_state=0, tol=0.0001, verbose=0)

# predict the closest cluster each sample in X belongs to.
y_kmeans = kmeans.predict(X)

# plot clustered labels
plt.scatter(X[:, 0], X[:, 1], c=y_kmeans, cmap='viridis')

# plot cluster centers
centers = kmeans.cluster_centers_
plt.scatter(centers[:, 0], centers[:, 1], c='black', s=200, alpha=0.7);
plt.show()
```