Kmeans Clustering in Sklearn

A Brief Walkthrough by DJ Nigh

Data Scientists have many tools in their tool belts. When modeling, those tools can generally fall into two categories: supervised and unsupervised learning. So far in Flatiron, only supervised learning has been covered. This type of learning is useful for classification and regression models. Essentially, you are trying to predict something: a value or a category. Let's focus on classification. Classification models take in labeled data with a known target. The model learns what is predictive of each target class based on the training data set. As previously mentioned, a litany of tools exist to this end, such as Random Forests or Logistic Regression. However, we don't always know what we are looking for.

For datasets where the target is either not obvious or unknown, unsupervised learning is critical. While more difficult to grasp, these models are widely used in the world around us. One of the most versatile modeling methods is K-means clustering. Clustering is a class of unsupervised learning designed to group data points together based on similarities. The model works like the supervised model, K-Nearest Neighbor. The data scientists select the number of clusters, or groups, that they would like as an output. The model then selects *k* points at random and calculates the distance of each value from each of the centroids. The points are sorted into groups to minimize intragroup variance. The model doesn't know if the originally guessed points are great centers for the group. As such, the model will iterate many times, selecting new centroids each time. Once a specified or default number of attempts have been made, the model selects the iterations with the lowest mean intragroup variance. While difficult to do by hand, the process is made quite easy with the Python library Sklearn.

When using Sklearn's sklearn.cluster.KMeans package, almost all of the defaults can generally be used and the user will only need to specify *k*. Great care is need when selecting k as the model does not think for itself. If k is 100 and the length of the data is also 100, the output would be 100 groups of a single point – likely not very useful. The same is true for under-binning: values that should not go together will be shoved together in order to match *k*. The best methodology is to map the mean intra-group variance for each *k*, selecting the *k* where variance ceases to rapidly decline. This is called the elbow, and the plot is called an elbow chart.

Let's walk through a quick example:

Take the following dataset of beak lengths (mm) of imaginary finches from various species:

- 10.0, 12.0, 13.3, 15.0, 16.3, 18.2, 19.5, 20.0

How many species are there? From a quick look, you could probably guess three. Where at? Once you've answer, let's see if the computer agrees.

Start by importing the proper packages:

```
import pandas as pd
from sklearn.cluster import KMeans
```

✓ 0.7s Python

Once you've done this, make the dataset useable with Pandas' Series function and reshape method.

```
beaks = pd.Series([10.0, 12.0, 13.3, 15.0, 16.3, 18.2, 19.5, 20.0], name= 'beak_length').values.reshape(-1,1)
```
✓ 0.0s Python

Instantiate the KMeans model with the proper hyperparameters. In this instance, random state was used to ensure repeatability for those following along.

```
clusterer = KMeans(n_clusters=3, random_state=10)
```
✓ 0.0s Python

Use the .fit_predict method to model the data and predict classes for the data points. The output is the predicted category for that value.

```
clusterer.fit_predict(beaks)
```
✓ 0.0s Python

```
array([0, 0, 0, 2, 2, 1, 1, 1])
```

If interested in the mean of each group (the centroid), that information is found using the .cluster_centers method.

```
clusterer.cluster_centers_
```
✓ 0.0s Python

```
array([[11.76666667],
       [19.23333333],
       [15.65      ]])
```

The output binning each passes the smell test for where the centers are, and which members belong to which group. While this is an easy data set to check visually, many will not be, and validation will need to be done.

In summary, data scientists will often work with data that does not a desired endpoint, otherwise known as unsupervised learning. One of the most popular issues they will have to handle is clustering, or predicting group membership. Kmeans clustering excels at this task. This machine learning model allows one to group data based on vector distance from a generated centroid. This tool has many applications from cybersecurity to biological taxonomy. In this instance, it was used to predict the species based on beak length. While simple, the example demonstrates how powerful Kmeans clustering would be on data sets too large for human computation.