K-Means Clustering Report

K-means clustering is a type of unsupervised learning, which is used when you have unlabeled data (i.e., data without defined categories or groups). The goal of this algorithm is to find groups in the data, with the number of groups represented by the variable *K*. The algorithm works iteratively to assign each data point to one of *K* groups based on the features that are provided. Data points are clustered based on feature similarity. The results of the *K*-means clustering algorithm are:

1. The centroids of the *K* clusters, which can be used to label new data
2. Labels for the training data (each data point is assigned to a single cluster)

Rather than defining groups before looking at the data, clustering allows you to find and analyze the groups that have formed organically. The "Choosing K" section below describes how the number of groups can be determined.

Each centroid of a cluster is a collection of feature values which define the resulting groups. Examining the centroid feature weights can be used to qualitatively interpret what kind of group each cluster represents.

This introduction to the *K*-means clustering algorithm covers:

- Common business cases where K-means is used
- The steps involved in running the algorithm
- A Python example using delivery fleet data

Algorithm

The K-means clustering algorithm uses iterative refinement to produce a final result. The algorithm inputs are the number of clusters K and the data set. The data set is a collection of features for each data point. The algorithms starts with initial estimates for the K centroids, which can either be randomly generated or randomly selected from the data set. The algorithm then iterates between two steps:

1. Data assigment step:

Each centroid defines one of the clusters. In this step, each data point is assigned to its nearest centroid, based on the squared Euclidean distance. More formally, if c_i is the collection of centroids in set C, then each data point x is assigned to a cluster based on

$$\underset{c_i \in C}{\arg\min}\ dist(c_i,\ x)^2$$

where $dist(\cdot)$ is the standard (L_2) Euclidean distance. Let the set of data point assignments for each i^{th} cluster centroid be S_i.

2. Centroid update step:

In this step, the centroids are recomputed. This is done by taking the mean of all data points assigned to that centroid's cluster.

$$c_i = \frac{1}{|S_i|}\sum_{x_i \in S_i} x_i$$

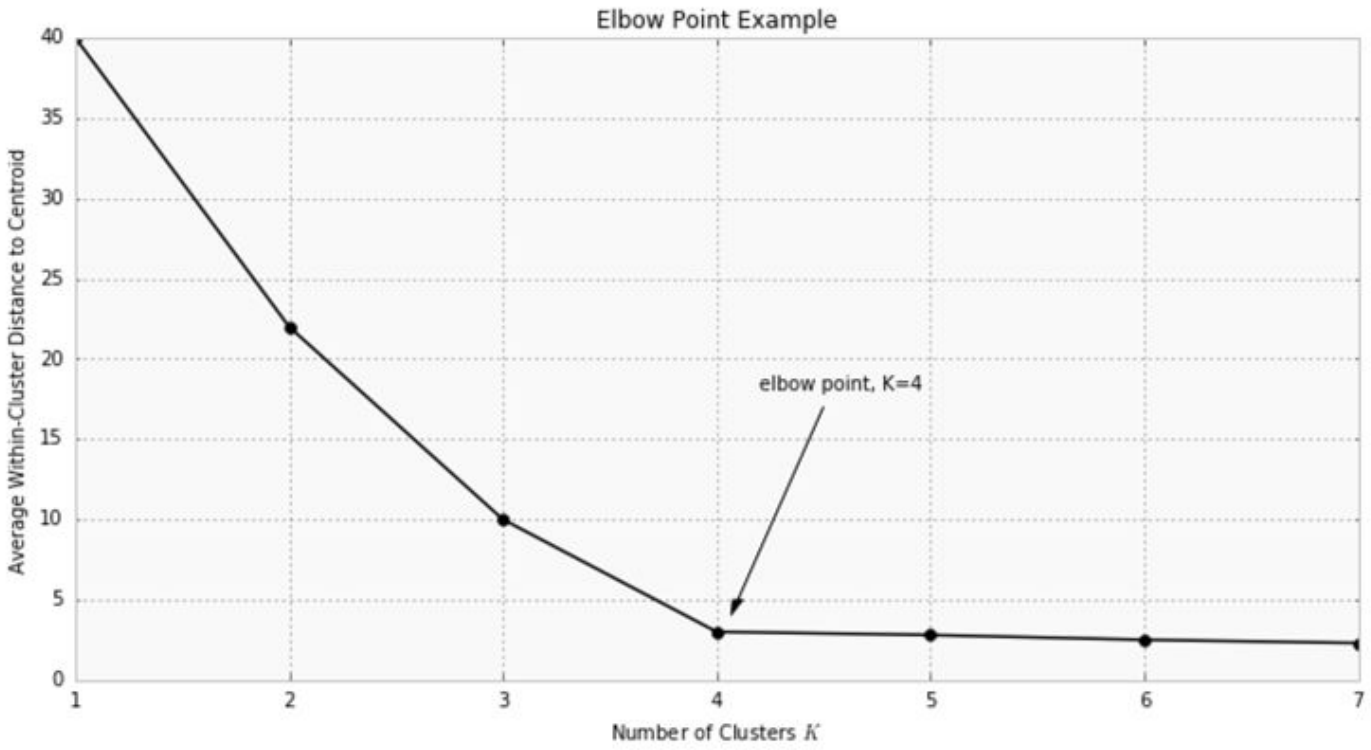
The algorithm iterates between steps one and two until a stopping criteria is met (i.e., no data points change clusters, the sum of the distances is minimized, or some maximum number of iterations is reached).

This algorithm is guaranteed to converge to a result. The result may be a local optimum (i.e. not necessarily the best possible outcome), meaning that assessing more than one run of the algorithm with randomized starting centroids may give a better outcome.

Choosing K

The algorithm described above finds the clusters and data set labels for a particular pre-chosen *K*. To find the number of clusters in the data, the user needs to run the *K*-means clustering algorithm for a range of *K* values and compare the results. In general, there is no method for determining exact value of *K*, but an accurate estimate can be obtained using the following techniques.
One of the metrics that is commonly used to compare results across different values of *K* is the mean distance between data points and their cluster centroid. Since increasing the number of clusters will always reduce the distance to data points, increasing *K* will *always* decrease this metric, to the extreme of reaching zero when *K* is the same as the number of data points. Thus, this metric cannot be used as the sole target. Instead, mean

distance to the centroid as a function of *K* is plotted and the "elbow point," where the rate of decrease sharply shifts, can be used to roughly determine *K*.
A number of other techniques exist for validating *K*, including cross-validation, information criteria, the information theoretic jump method, the silhouette method, and the G-means algorithm. In addition, monitoring the distribution of data points across groups provides insight into how the algorithm is splitting the data for each *K*.



Example: Applying K-Means Clustering to Delivery Fleet Data

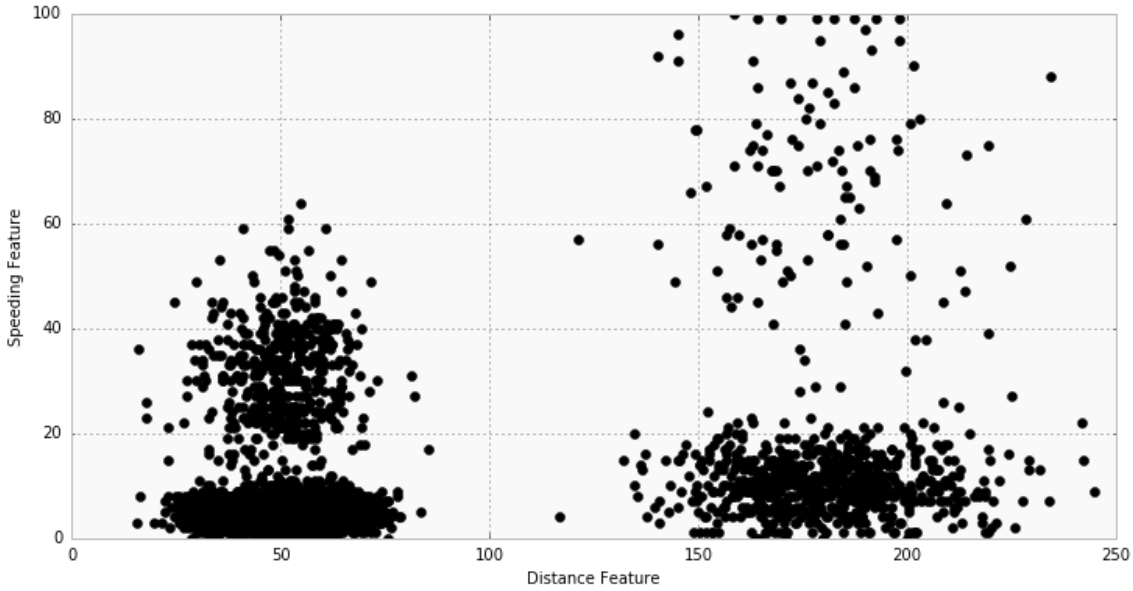
As an example, we'll show how the *K*-means algorithm works with a sample dataset of delivery fleet driver data. For the sake of simplicity, we'll only be looking at two driver features: mean distance driven per day and the mean percentage of time a driver was >5 mph over the speed limit. In general, this algorithm can be used for any number of features, so long as the number of data samples is much greater than the number of features.

Step 1: Clean and Transform Your Data

For this example, we've already cleaned and completed some simple data transformations. A sample of the data as a pandas DataFrame is shown below.

	Driver_ID	Distance_Feature	Speeding_Feature
0	3423311935	71.24	28
1	3423313212	52.53	25
2	3423313724	64.54	27
3	3423311373	55.69	22
4	3423310999	54.58	25

The chart below shows the dataset for 4,000 drivers, with the distance feature on the x-axis and speeding feature on the y-axis.



Step 2: Choose K and Run the Algorithm

Start by choosing *K*=2. For this example, use the Python packages scikit-learn and NumPy for computations as shown below:

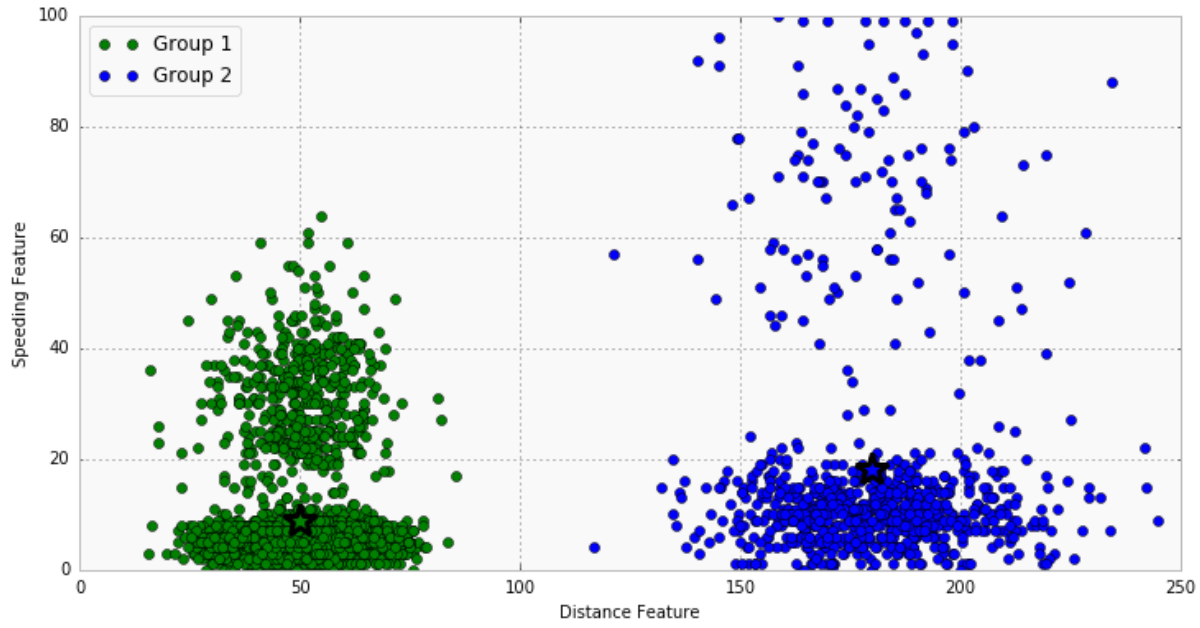
```
import numpy as np
from sklearn.cluster import KMeans

### For the purposes of this example, we store feature data from our
### dataframe `df`, in the `f1` and `f2` arrays. We combine this
into
### a feature matrix `X` before entering it into the algorithm.
f1 = df['Distance_Feature'].values
f2 = df['Speeding_Feature'].values

X=np.matrix(zip(f1,f2))
kmeans = KMeans(n_clusters=2).fit(X)
```

The cluster labels are returned in kmeans.labels_.

Step 3: Review the Results The chart below shows the results. Visually, you can see that the *K*-means algorithm splits the two groups based on the distance feature. Each cluster centroid is marked with a star.

- Group 1 Centroid = (50, 5.2)
- Group 2 Centroid = (180.3, 10.5)

Using domain knowledge of the dataset, we can infer that Group 1 is urban drivers and Group 2 is rural drivers.



Step 4: Iterate Over Several Values of K

Test how the results look for *K*=4. To do this, all you need to change is the target number of clusters in the KMeans() function.

The chart below shows the resulting clusters. We see that four distinct groups have been identified by the algorithm; now speeding drivers have been separated from those who follow speed limits, in addition to the rural vs. urban divide. The threshold for speeding is lower with the urban driver group than for the rural drivers, likely due to urban drivers spending more time in intersections and stop-and-go traffic.