ORACLE + DATASCIENCE.COM

LEARN DATA SCIENCE, MACHINE LEARNING

Introduction to K-means Clustering

Author: Andrea Trevino / Posted on December 6, 2016

Prerequisites

- Experience with the specific topic: Novice

- Professional experience: No industry experience

Knowledge of machine learning is not required, but the reader should be familiar with basic data analysis (e.g., descriptive analysis) and the programming language Python. To follow along, download the sample dataset here.

Introduction To K-means Clustering

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

Business Uses

The *K*-means clustering algorithm is used to find groups which have not been explicitly labeled in the data. This can be used to confirm business assumptions about what types of groups exist or to identify unknown groups in complex data sets. Once the algorithm has been run and the groups are defined, any new data can be easily assigned to the correct group.

This is a versatile algorithm that can be used for any type of grouping. Some examples of use cases are:

- Behavioral segmentation:
    - Segment by purchase history
    - Segment by activities on application, website, or platform
    - Define personas based on interests
    - Create profiles based on activity monitoring
- Inventory categorization:
    - Group inventory by sales activity
    - Group inventory by manufacturing metrics
- Sorting sensor measurements:
    - Detect activity types in motion sensors
    - Group images
    - Separate audio
    - Identify groups in health monitoring
- Detecting bots or anomalies:
    - Separate valid activity groups from bots
    - Group valid activity to clean up outlier detection

In addition, monitoring if a tracked data point switches between groups over time can be used to detect meaningful changes in the

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.

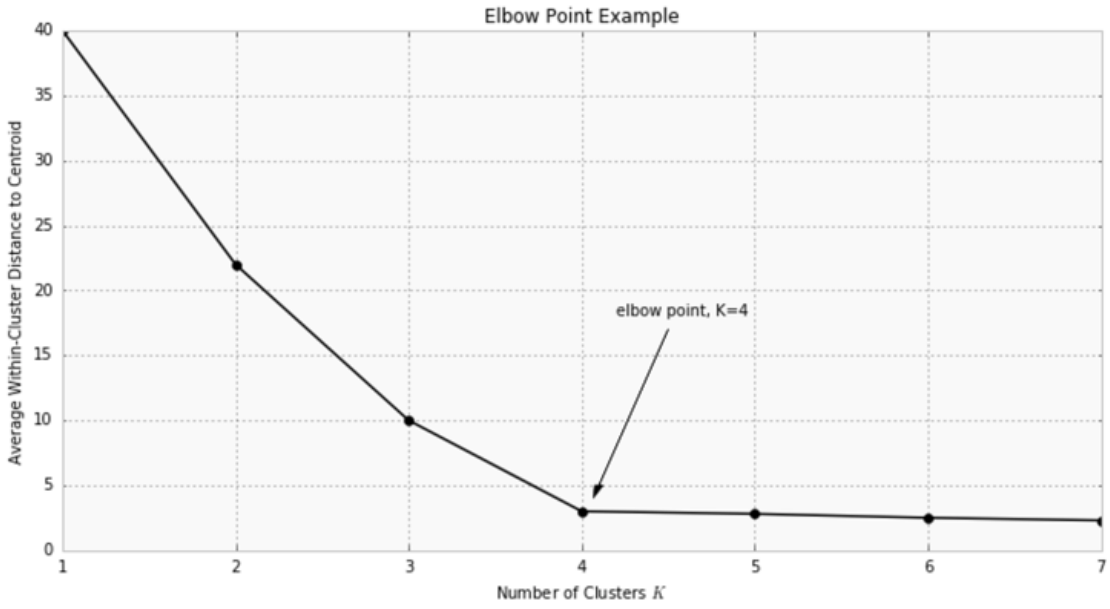
Choosing K

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

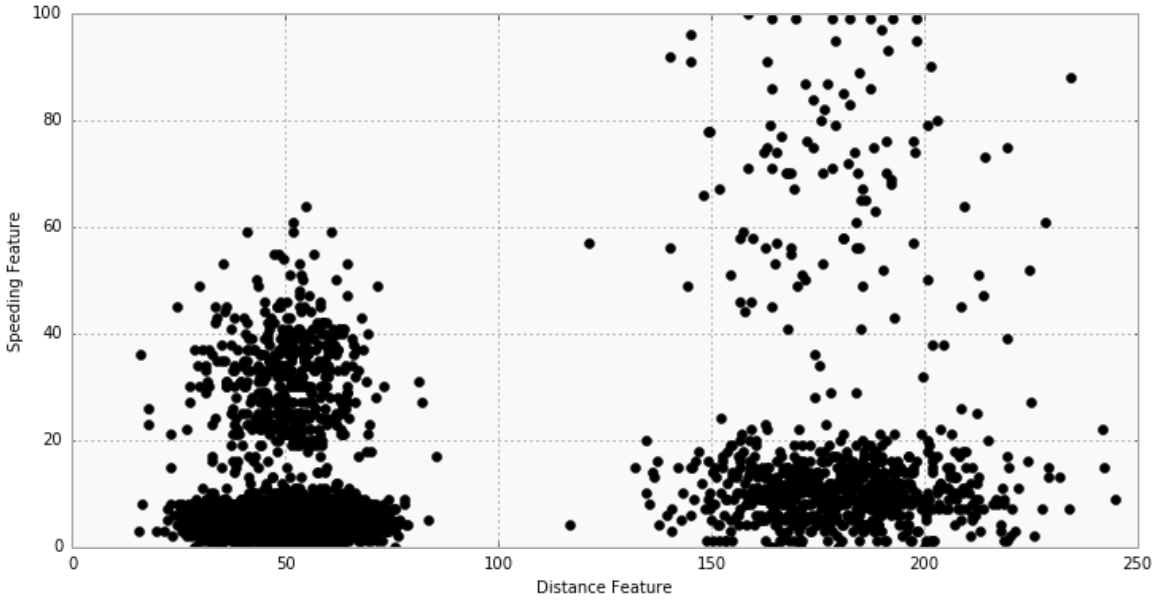
As an example, we'll show how the *K*-means algorithm works with a sample dataset of delivery fleet driver data. For the sake of simplicity, we'll only be looking at two driver features: mean distance driven per day and the mean percentage of time a driver was >5 mph over the speed limit. In general, 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)
```

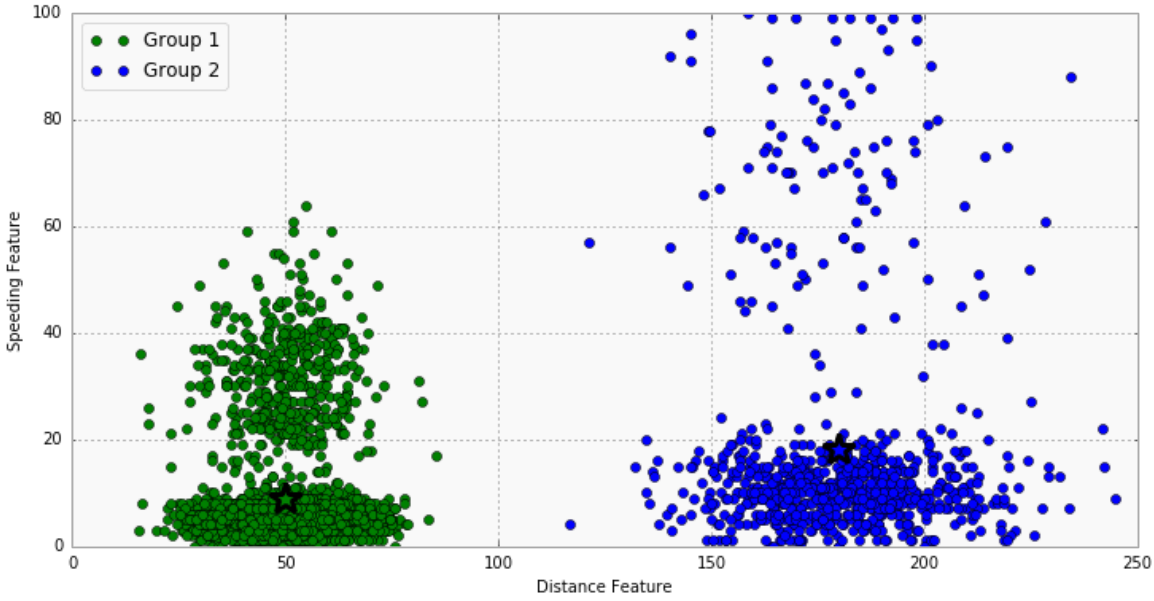
The cluster labels are returned in **`kmeans.labels_`**.

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.

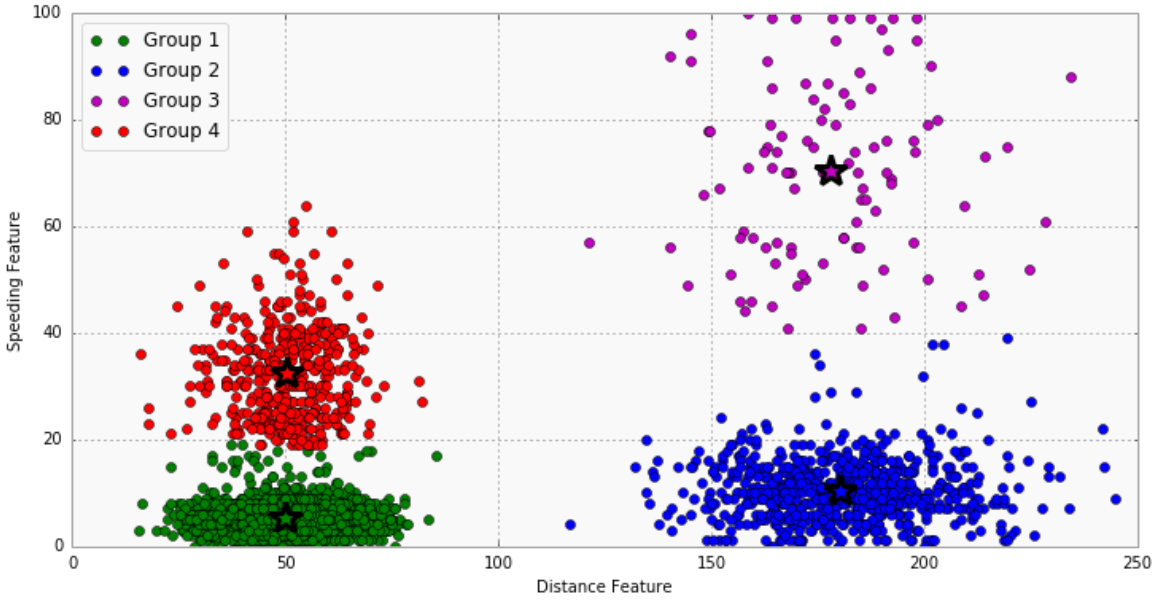


Step 4: Iterate Over Several Values of K

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.

```
kmeans = KMeans(n_clusters=4).fit(X)
```

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.



Additional Notes And Alternatives

Feature Engineering

Feature engineering is the process of using domain knowledge to choose which data metrics to input as features into a machine learning algorithm. Feature engineering plays a key role in *K*-means clustering; using meaningful features that capture the variability of the data is essential for the algorithm to find all of the naturally-occurring groups.

Categorical data (i.e., category labels such as gender, country, browser type) needs to be encoded or separated in a way that can

still work with the algorithm.

Feature transformations, particularly to represent rates rather than measurements, can help to normalize the data. For example, in the delivery fleet example above, if total distance driven had been used rather than mean distance per day, then drivers would have been grouped by how long they had been driving for the company rather than rural vs. urban.

Alternatives

A number of alternative clustering algorithms exist including DBScan, spectral clustering, and modeling with Gaussian mixtures. A dimensionality reduction technique, such as principal component analysis, can be used to separate groups of patterns in data. You can read more about alternatives to K-means in this post.

One possible outcome is that there are no organic clusters in the data; instead, all of the data fall along the continuous feature ranges within one single group. In this case, you may need to revisit the data features to see if different measurements need to be included or a feature transformation would better represent the variability in the data. In addition, you may want to impose categories or labels based on domain knowledge and modify your analysis approach.

For more information on K-means clustering, visit the

scikit learn site.

Want to keep learning? Download our new study from Forrester about the tools and practices keeping companies on the forefront of data science.

Author Andrea Trevino

Data Scientist. Loves unravelling data puzzles and will spend hours talking about anything related to audio, languages, or food.

Enjoyed this post? Don't forget to share.

Related Content

LEARN DATA SCIENCE

Transfer Learning in PyTorch, Part 2: How to Create a Transfer Learning Class and Train on Kaggle's Test Set

LEARN MORE ›

LEARN DATA SCIENCE

Transfer Learning in PyTorch, Part 1: How to Use DataLoaders and Build a Fully Connected Class

LEARN MORE ›

LEARN DATA SCIENCE

4 Types of Data Analytics

LEARN MORE ›

LEARN DATA SCIENCE

4 Approaches to Overcoming Label Bias in Positive and Unlabeled Learning

LEARN MORE ›

SUBSCRIBE TO OUR NEWSLETTER →