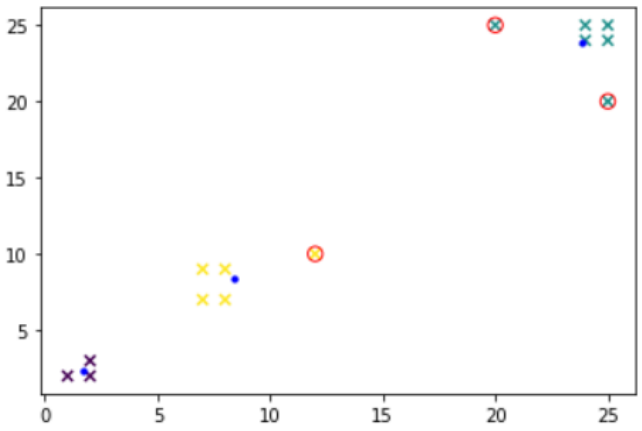
Outlier Detection with K-means Clustering in Python

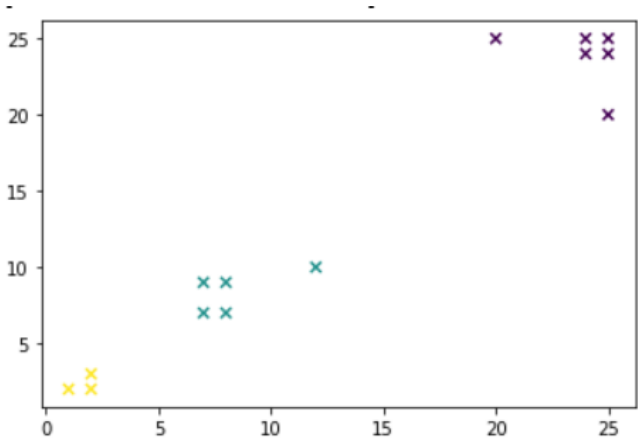
Detecting outliers using k-means clustering explained in a very simple form.

A. Kübra Kuyucu Follow
Feb 1 · 3 min read ★



Data with outliers detected by Author

K-means clustering is used when you want to cluster your data into k groups. I will tell you how to catch the outliers that stay far away from these groups. We will do it by deciding a threshold ratio. For each cluster, the data stay out the threshold ratio will be counted as an outlier.



Data to be used by Author

When you look at the plot, it is easier to see which points we aim to catch. In the yellow

...usters
respectively. So, we aim to catch three outliers in this data set.

We first import the necessary libraries and compose the data. Then, the k-means clusters predicted by setting k = 3. Lastly, we get the plot above by running this code.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from scipy.spatial.distance import cdist

# composing data set
data = np.array([[1, 2], [2, 2], [2, 3], [8, 7], [8, 9], [7, 9], [7,
7], [12,10], [25, 24], [24, 24], [24, 25], [25, 25], [25,20],
[20,25]])

# kmeans model, setting k = 3
km = KMeans(n_clusters = 3)
clusters=km.fit_predict(data)

# plotting data set
plt.scatter(*zip(*data),c=clusters,marker = "x")
```

Now, we will find the centers of clusters and then calculate the distances between each point to the centers of its cluster.

```
# obtaining the centers of the clusters
centroids = km.cluster_centers_

# points array will be used to reach the index easy
points = np.empty((0,len(data[0])), float)

# distances will be used to calculate outliers
distances = np.empty((0,len(data[0])), float)

# getting points and distances
for i, center_elem in enumerate(centroids):
    # cdist is used to calculate the distance between center and
other points
    distances = np.append(distances,
cdist([center_elem],data[clusters == i], 'euclidean'))
    points = np.append(points, data[clusters == i], axis=0)
```

You may ask which algorithm do we use to calculate the distances and can we choose any other one. As you may recognize, in the *cdist* function, we give distance type as a parameter which is 'euclidean'. You can replace it with any other one that *cdist accepts*.

After obtaining the distances of points, now, we will decide a threshold ratio as a percentile and find the outliers.

When you decide on a threshold ratio *th*, you are sorting all distances of all points (to their own centers) and then saying that I want the points to be outliers that are above percentile *th*. I set it to 80 but you can play around with it.

```
percentile = 80
# getting outliers whose distances are greater than some percentile
```

So, we are done! The only thing we left is to visualize our data with outliers detected.

```
fig = plt.figure()

# plotting initial data
plt.scatter(*zip(*data),c=clusters,marker = "x")

# plotting red ovals around outlier points
plt.scatter(*zip(*outliers),marker="o",facecolor="None",edgecolor="r"
,s=70);

# plotting centers as blue dots
plt.scatter(*zip(*centroids),marker="o",facecolor="b",edgecolor="b",s
=10);
```


Data with outliers detected by Author

The blue points in the plot represent the center of clusters. The cluster colors have changed but it isn't important. The outliers are signed with red ovals.

If you want to use this algorithm to detect outliers that are staying out of all data but not clusters, you need to choose k = 1.

```
# setting k = 1
km = KMeans(n_clusters = 1)
```

Data Science Outliers Python Clustering Artificial Intelligence