Clustering using k-means

Contents


```
import numpy as np
np.set_printoptions(precision=3)
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
sns.set(rc={"figure.dpi":100, "savefig.dpi":300})
sns.set_context("notebook")
sns.set_style("ticks")
```

Objectives

- To demonstrate clustering using k-means

Let's start by generating a synthetic dataset using with three clusters:

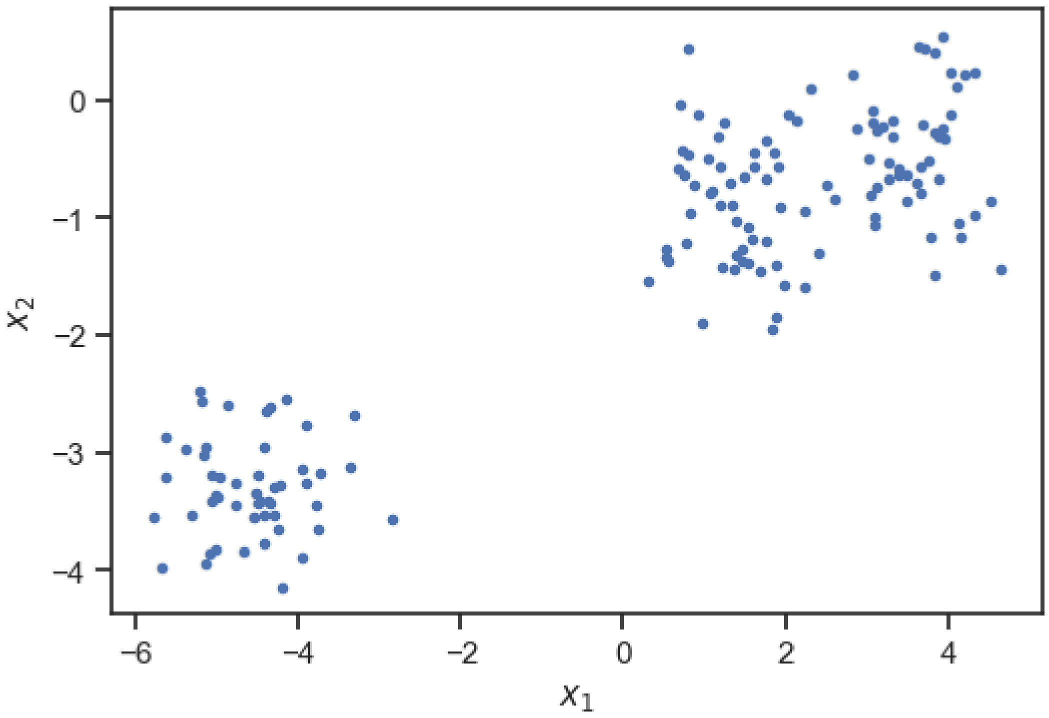
```
np.random.seed(123456)

# Make synthetic dataset for clustering
num_clusters_true = 3
# The means of each cluster
mu_true = 3.0 * np.random.randn(num_clusters_true, 2)
# The variance of the observations around the cluster
sigma_true = 0.5
# How many observations to generate per cluster
num_obs_cluster = [50, 50, 50]

# Generate the data
data = []
for i in range(num_clusters_true):
    x_i = mu_true[i] + sigma_true * np.random.randn(num_obs_cluster[i], 2)
    data.append(x_i)
data = np.vstack(data)
# Permute the data so that order info is lost
data = np.random.permutation(data)
```

Now let's visualize the data forgetting about the underlying clusers that gave rise to them.

```
fig, ax = plt.subplots()
ax.plot(data[:, 0], data[:, 1], '.')
ax.set_xlabel('$x_1$')
ax.set_ylabel('$x_2$');
```



We are not going to implement the K-means algorithm from scratch. Instead, we are going to use the implementation that can be found in [scikit-learn](). Here is how easy it is:

```
from sklearn.cluster import KMeans

model = KMeans(n_clusters=3).fit(data)
```

Here is how you can access the cluster centers (the μ_k's) from the trained model:

```
model.cluster_centers_
```

```
array([[ 3.61 , -0.443],
       [-4.572, -3.297],
       [ 1.394, -0.89 ]])
```

Compare the identified cluster centers to the actual cluster centers:

```
mu_true
```

```
array([[ 1.407, -0.849],
       [-4.527, -3.407],
       [ 3.636, -0.52 ]])
```

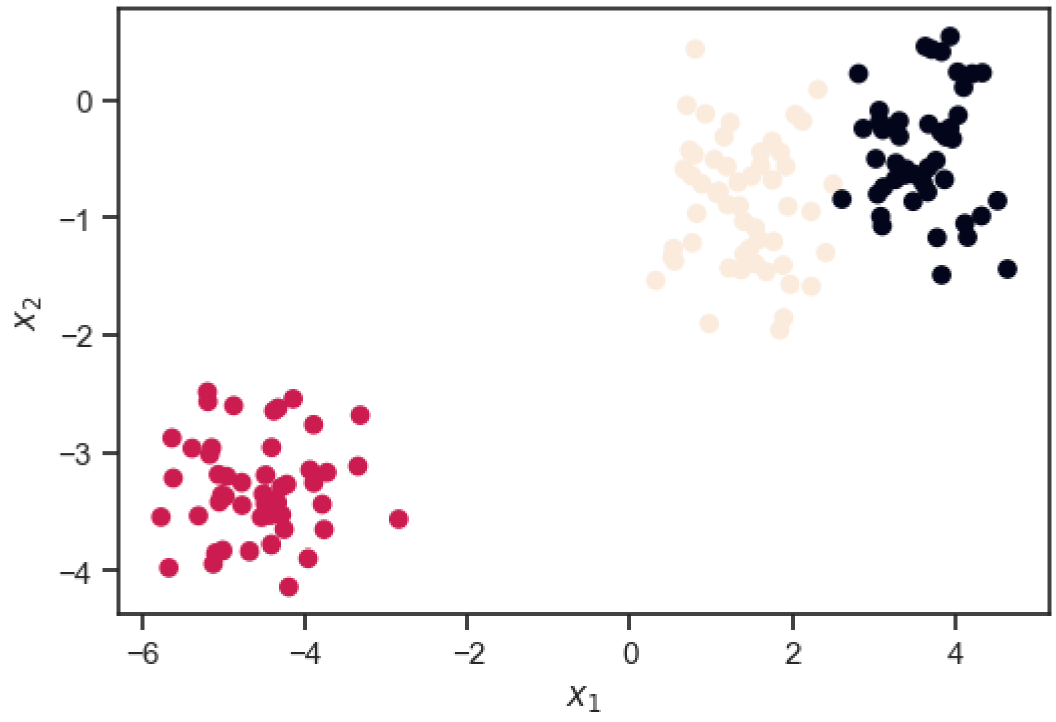
K-means has also labeled each observation point with its cluster id. Here is how to get this info:

```
model.labels_
```

```
array([2, 2, 1, 1, 1, 1, 0, 1, 2, 0, 0, 2, 1, 1, 2, 0, 0, 2, 1, 1, 0, 0,
       1, 1, 2, 2, 0, 2, 2, 2, 0, 0, 2, 1, 0, 2, 1, 2, 0, 0, 2, 0, 2, 0,
       0, 0, 1, 2, 1, 2, 0, 0, 1, 2, 1, 1, 2, 0, 2, 1, 2, 2, 0, 2, 0, 0,
       1, 0, 1, 1, 2, 2, 1, 1, 1, 2, 1, 0, 2, 1, 2, 2, 0, 1, 2, 1, 1, 2,
       0, 0, 2, 2, 0, 1, 2, 0, 2, 2, 0, 2, 0, 1, 1, 0, 1, 2, 1, 1, 0, 1,
       1, 2, 2, 2, 1, 2, 2, 1, 0, 2, 0, 1, 1, 0, 2, 0, 0, 1, 1, 1, 1, 1,
       1, 0, 0, 0, 2, 2, 0, 2, 0, 2, 1, 0, 2, 0, 2, 1, 2, 2], dtype=int32)
```

Since we have 2D observations, we can actually visualize the clusters. Here is a nice way to do this:

```
labels = model.predict(data)
fig, ax = plt.subplots()
plt.scatter(data[:, 0], data[:, 1], c=labels)
ax.set_xlabel('$x_1$')
ax.set_ylabel('$x_2$');
```

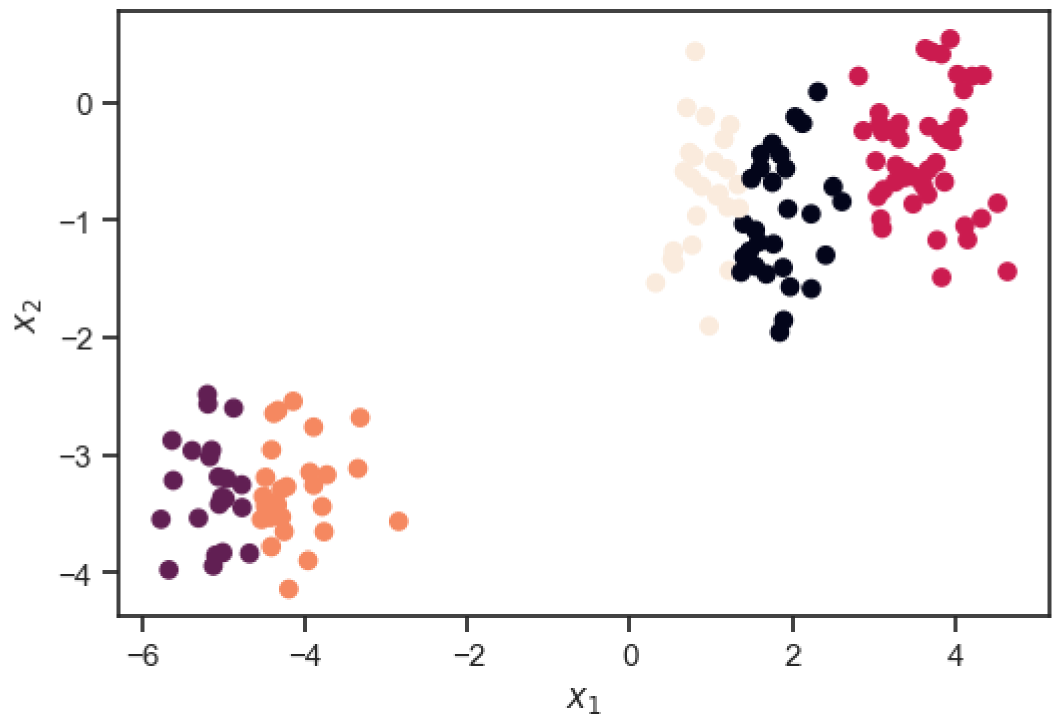


Okay, this seems to work perfectly. However, notice that we asked K-means to find three clusters which happens to be the true number of clusters in our dataset. What would happen if we had asked K-means to find a larger number of clusters, say 5? Here it is:

```
model5 = KMeans(n_clusters=5).fit(data)

labels = model5.predict(data)

fig, ax = plt.subplots()
plt.scatter(data[:, 0], data[:, 1], c=labels)
ax.set_xlabel('$x_1$')
ax.set_ylabel('$x_2$');
```



Questions

- We saw what happens when you ask K-means to find more clusters than there actually exist. What would happen, if you asked it to find a smaller number of clusters? Try $K=1$ and $K=2$ in the code block immediatly above. What do you observe? Can choose between $K=1,2,$ or 3?
- Rerun the entire example from the very first code block but this time set the number of true clusters to 6. Investigate what happens when you try to fit K-means with a very small number of clusters, what happens when you pick K to be around 6, and what happens when you pick a very big K, say 10.

By Ilias Bilionis (ibilion[at]purdue.edu)