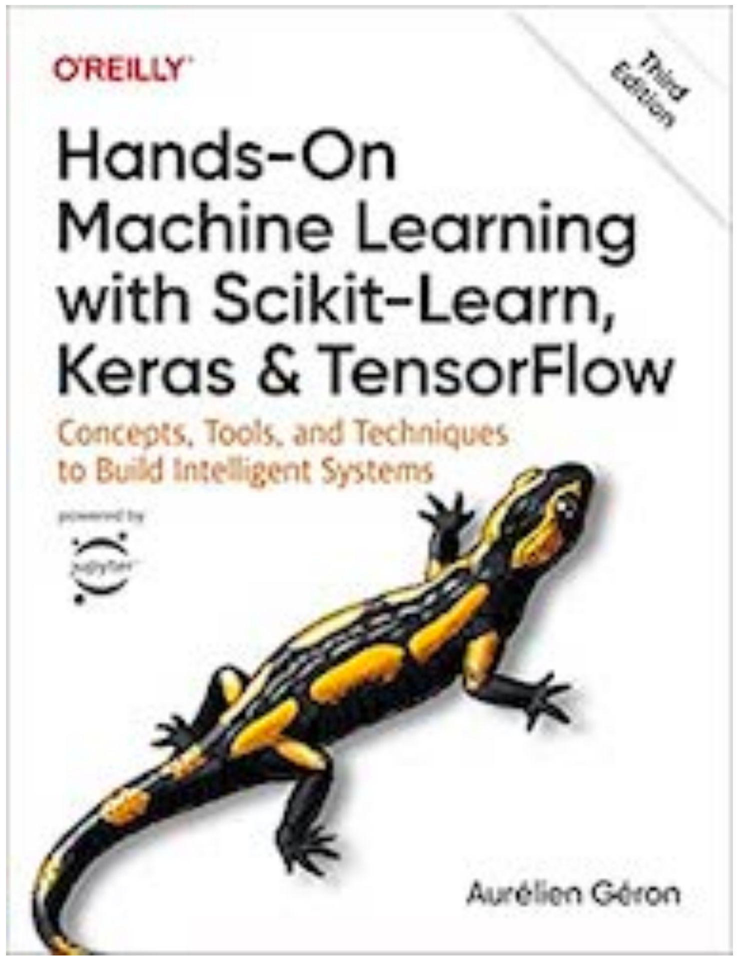
Machine Learning Security

9 Unsupervised Learning Techniques


O'REILLY
Third Edition
Hands-On Machine Learning with Scikit-Learn, Keras & TensorFlow
Concepts, Tools, and Techniques to Build Intelligent Systems
Aurélien Géron

Made Oct 15, 2023

Unsupervised Learning

- Most data is unlabeled
- Labeling usually requires human workers
- Unsupervised learning tasks:
  - Dimensionality reduction
  - Clustering
  - Anomaly detection
  - Density estimation
    - Determining the *probability density function* of the data

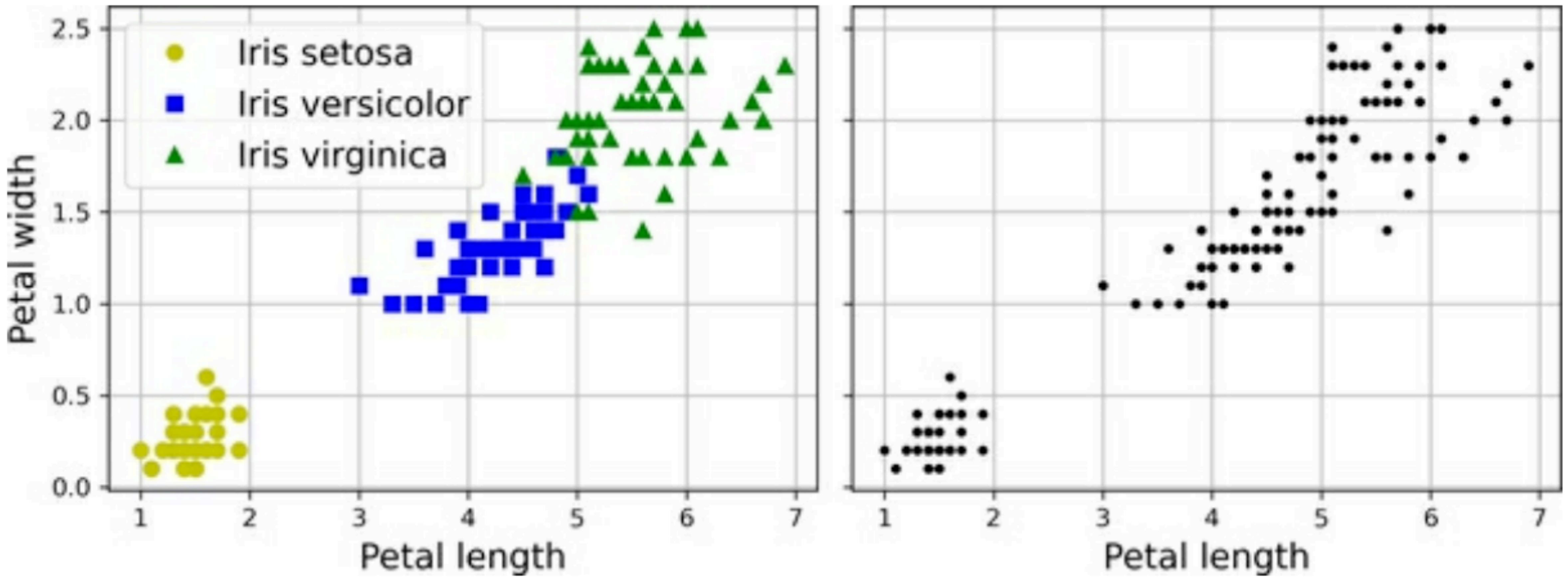
Topics

- Clustering Algorithms: k-Means and DBSCAN
- Gaussian Mixtures

Clustering Algorithms: k-Means and DBSCAN

Classification v. Clustering

- Classification on the left
  - Requires labels
- Clustering on the right
  - Can make two clusters, not three, from this data



Clustering Use Cases

- Customer segmentation
  - Grouping customers into clusters
  - Adapt products, marketing, and recommendations to them
- Data analysis
  - Group data into clusters, analyze clusters separately
- Dimensionality reduction
  - Find ***k*** clusters
  - Measure each instance's ***affinity*** with each cluster
  - Replace instance with its cluster affinities
  - This is ***k***-dimensional

Clustering Use Cases

- Feature engineering
  - Cluster affinities may be useful as extra features
- Anomaly detection
  - Instances with low affinity to all clusters are anomalies
- Semi-supervised learning
  - Some data is labeled
  - Form clusters and propagate labels to all instances in the same cluster

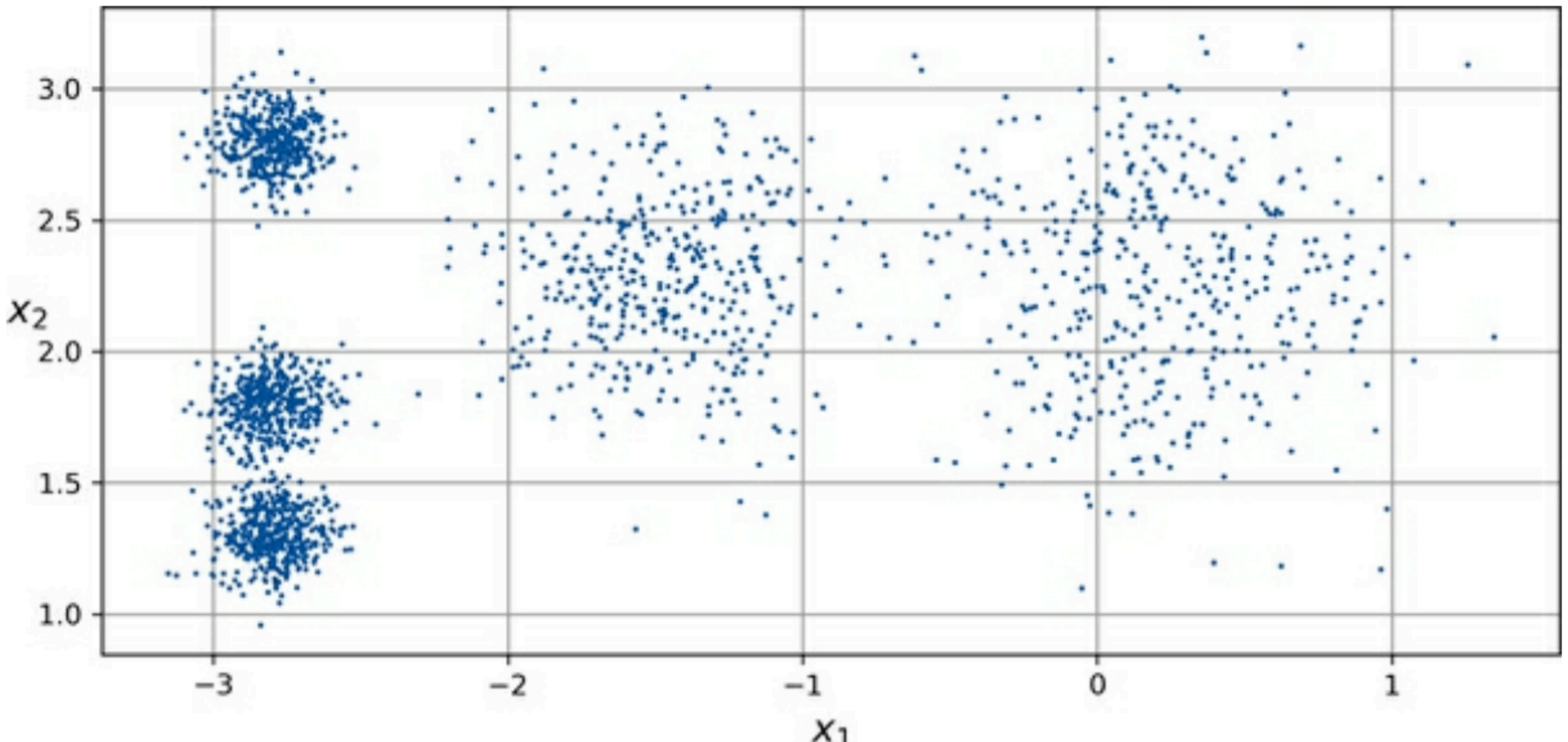
Clustering Use Cases

- Search engines
  - Search for images similar to a reference image
  - Cluster images, return the matching cluster
- Image segmentation
  - Cluster pixels by color

k-Means

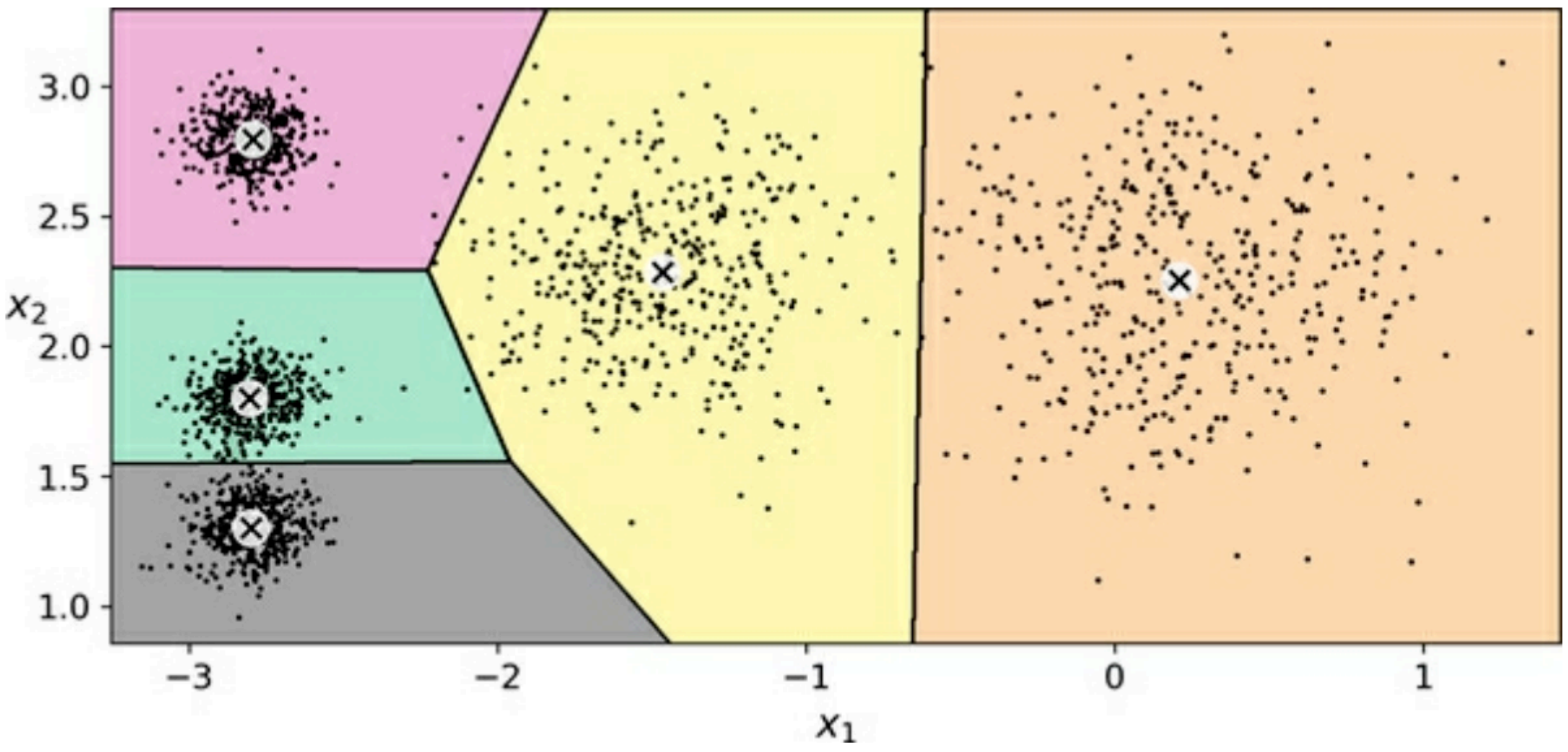
k-Means

- Proposed by Bell Labs in 1957
- This data has five Gaussian blobs



k-Means Decision Boundaries

- Minimizes distances to the centroids
- Converges to a good solution when the blobs all have the same size
- Note errors near the top left decision boundary



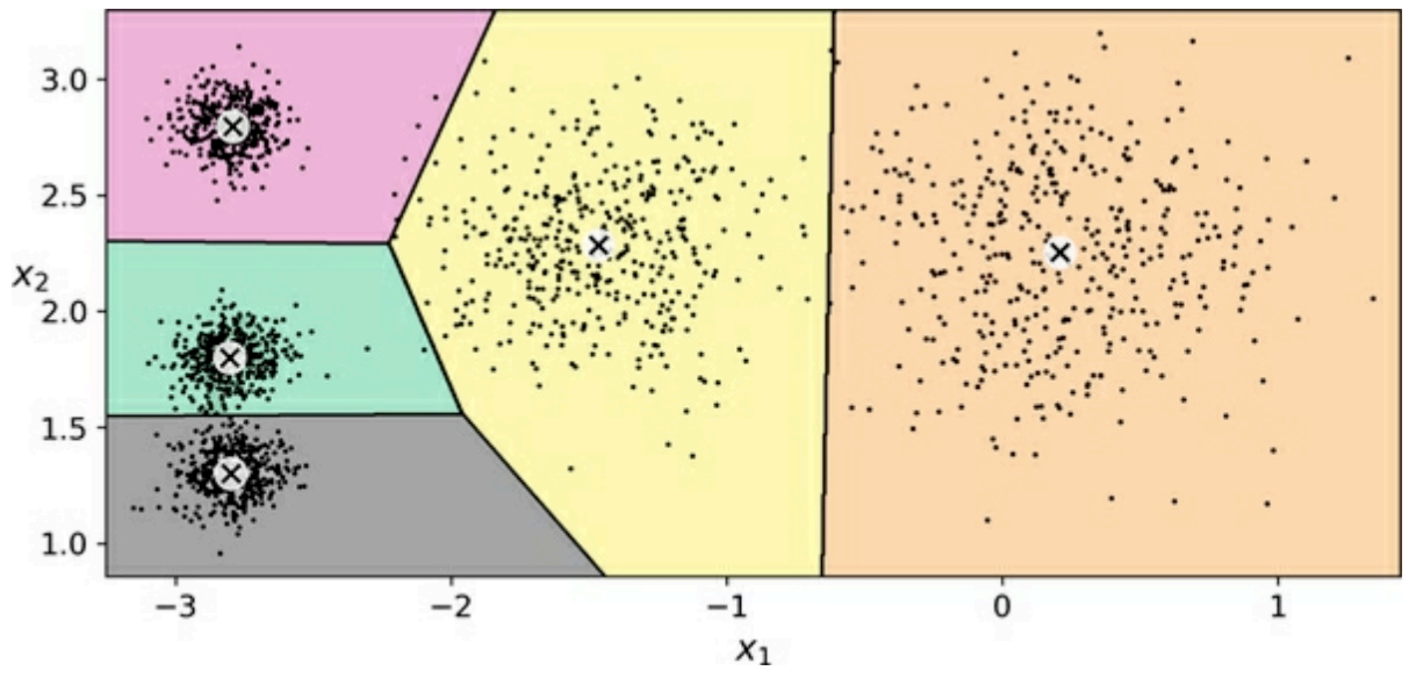
Hard v Soft Clustering

- Hard clustering
    - Assign each instance to a single cluster
- Soft clustering
    - Give each instance a score per cluster
    - Such as distance to the centroid

Dimensionality Reduction

- Original data was two-dimensional (x_1, x_2)
- Replace with 5-dimensional data, with distance from each instance to the centroids
- If original data has high dimensionality, this is a method of dimensionality reduction

```
>>> kmeans.transform(X_new).round(2)
array([[2.81, 0.33, 2.9 , 1.49, 2.89],
       [5.81, 2.8 , 5.85, 4.48, 5.84],
       [1.21, 3.29, 0.29, 1.69, 1.71],
       [0.73, 3.22, 0.36, 1.55, 1.22]])
```

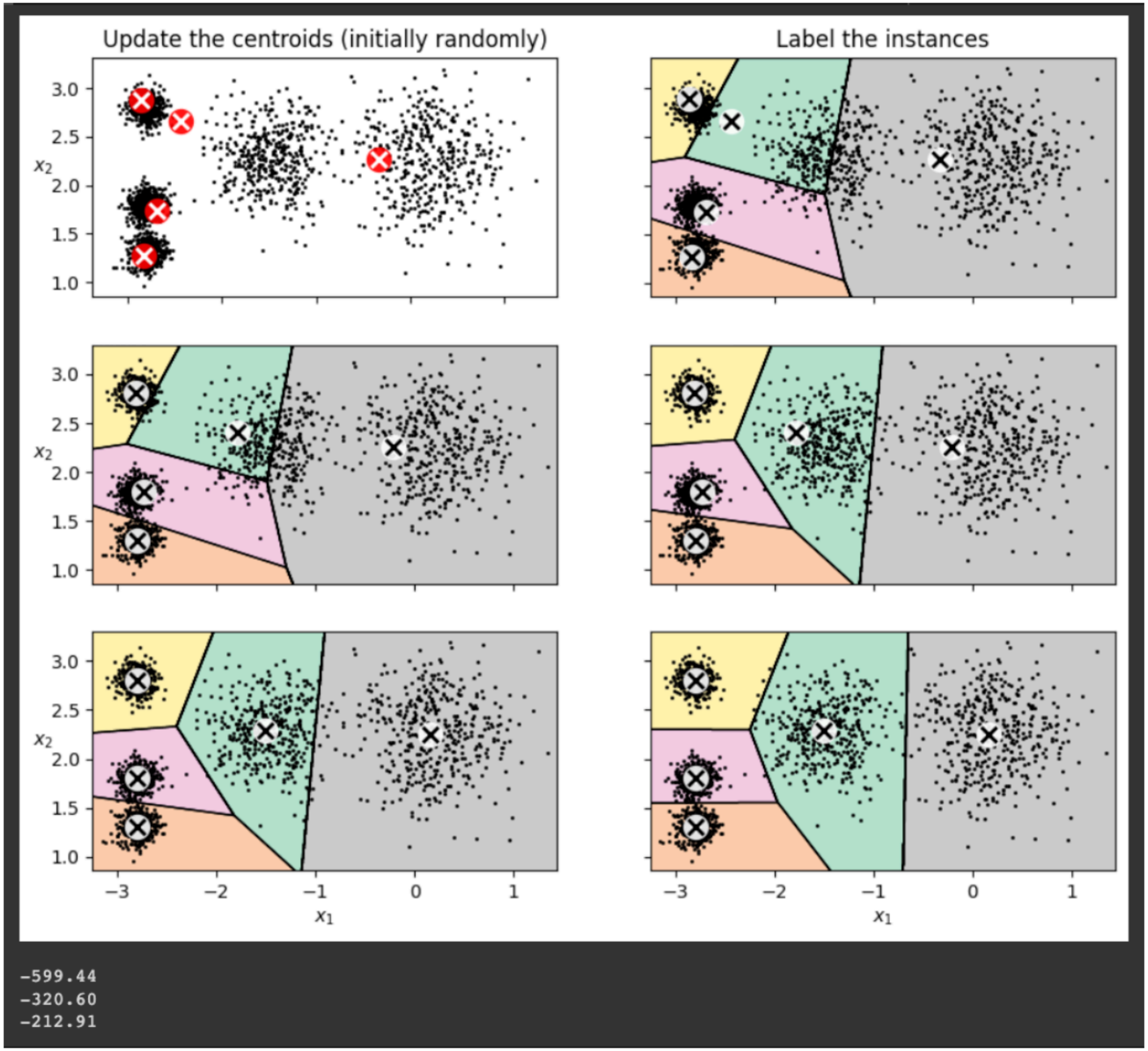


k-Means Algorithm

1. Starts by randomly placing the centroids
2. Label the instances by finding the nearest centroid
3. Updates the centroids to the center of the instances in that class
4. Loop back to step 2 for the next iteration

Scores are sum of squared distances from centroids, negated

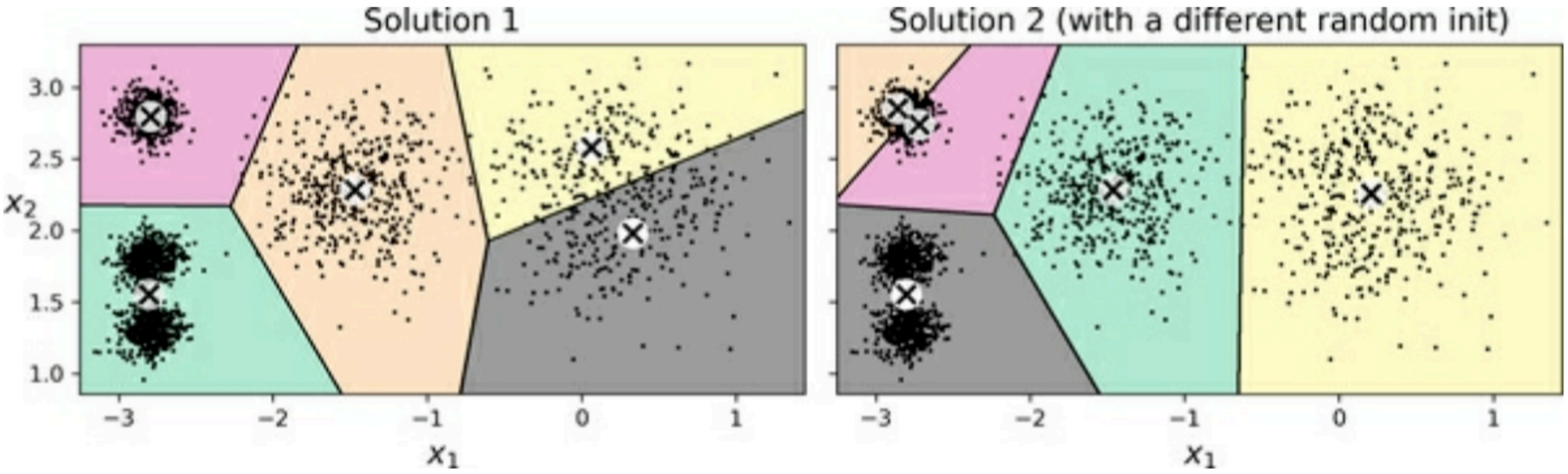
The positive score is called *inertia*



```
-599.44
-320.60
-212.91
```

Suboptimal Solutions

- Bad luck with initial centroids
- It tries **n_init** different starting locations (default 10)



Computational Complexity

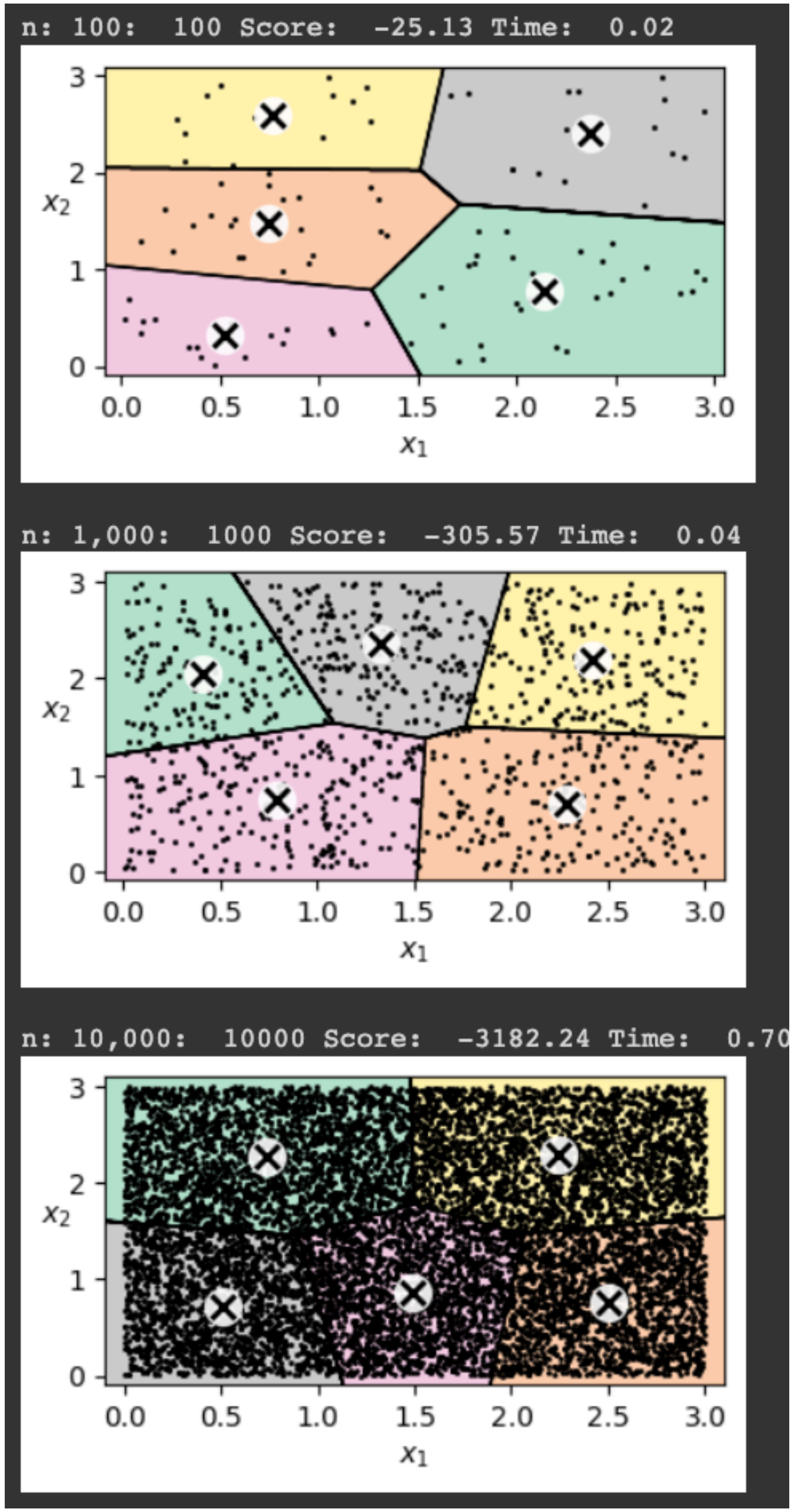
- If the data has clusters,
  - Linear in
    - Number of instances ***m***
    - Number of clusters ***k***
    - Number of dimensions ***n***
  - But if the data lacks a clustered structure, the model can fail to converge and increase in complexity exponentially
  - This rarely happens; in practice, **k-Means** is one of the fastest clustering algorithms

Random Data Part 1

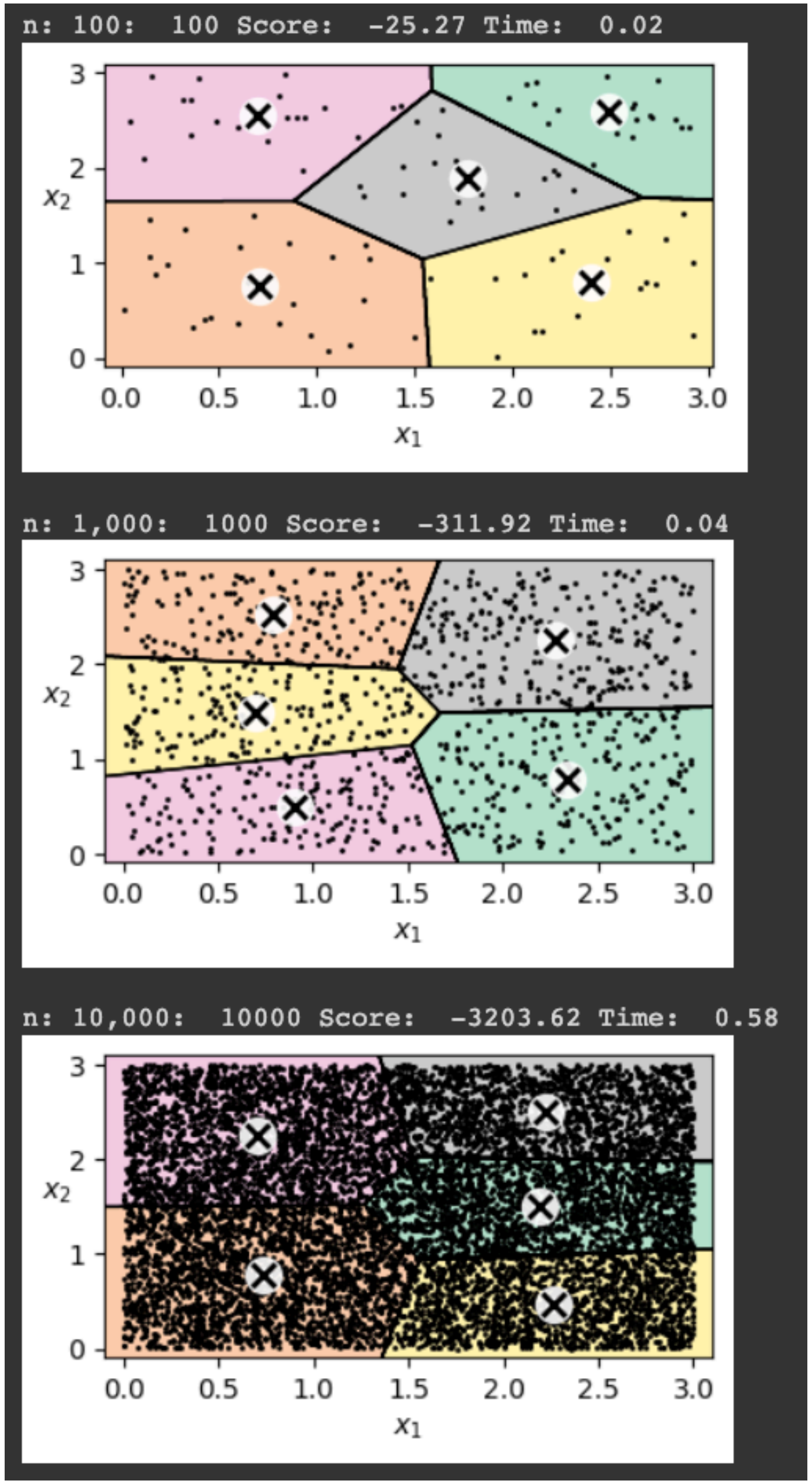
```
import numpy as np
import matplotlib.pyplot as plt
import random
from sklearn.cluster import KMeans
import time

def plot_data(X):
    plt.plot(X[:, 0], X[:, 1], 'k.', markersize=2)

def plot_centroids(centroids, weights=None, circle_color='w', cross_color='k'):
    if weights is not None:
        centroids = centroids[weights > weights.max() / 10]
    plt.scatter(centroids[:, 0], centroids[:, 1],
                marker='o', s=35, linewidths=8,
                color=circle_color, zorder=10, alpha=0.9)
    plt.scatter(centroids[:, 0], centroids[:, 1],
                marker='x', s=2, linewidths=12,
                color=cross_color, zorder=11, alpha=1)

def plot_decision_boundaries(clusterer, X, resolution=1000, show_centroids=True,
                             show_xlabels=True, show_ylabels=True):
    mins = X.min(axis=0) - 0.1
    maxs = X.max(axis=0) + 0.1
    xx, yy = np.meshgrid(np.linspace(mins[0], maxs[0], resolution),
                         np.linspace(mins[1], maxs[1], resolution))
    Z = clusterer.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)

    plt.contourf(Z, extent=(mins[0], maxs[0], mins[1], maxs[1]),
                cmap="Pastel2")
    plt.contour(Z, extent=(mins[0], maxs[0], mins[1], maxs[1]),
                linewidths=1, colors='k')
    plot_data(X)
    if show_centroids:
        plot_centroids(clusterer.cluster_centers_)

    if show_xlabels:
        plt.xlabel("$x_1$")
    else:
        plt.tick_params(labelbottom=False)
    if show_ylabels:
        plt.ylabel("$x_2$", rotation=0)
    else:
        plt.tick_params(labelleft=False)
```

Random Data
Part 2

```
for n in [100, 1_000, 10_000]:    # Number of data points

  # Make random data
  X = np.zeros(shape=(n,2))
  for i in range(n):
    X[i] = [random.uniform(0,3), random.uniform(0,3)]

  # Fit 5-way k-Means
  time_start = time.time()
  k = 5
  kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
  y_pred = kmeans.fit_predict(X)
  time_elapsed = time.time() - time_start

  print()
  print(f"n: {n:,}: ", n, "Score: ", "{:.2f}".format(kmeans.score(X)), "Time: ", "{:.2f}".format(time_elapsed))
  plt.figure(figsize=(4, 2))
  plot_decision_boundaries(kmeans, X)
  plt.show()
```



```
n: 100:   100 Score:   -25.13 Time:   0.02
n: 1,000:   1000 Score:   -305.57 Time:   0.04
n: 10,000:   10000 Score:   -3182.24 Time:   0.70
```



```
n: 100:   100 Score:   -25.27 Time:   0.02
n: 1,000:   1000 Score:   -311.92 Time:   0.04
n: 10,000:   10000 Score:   -3203.62 Time:   0.58
```

k-Means++

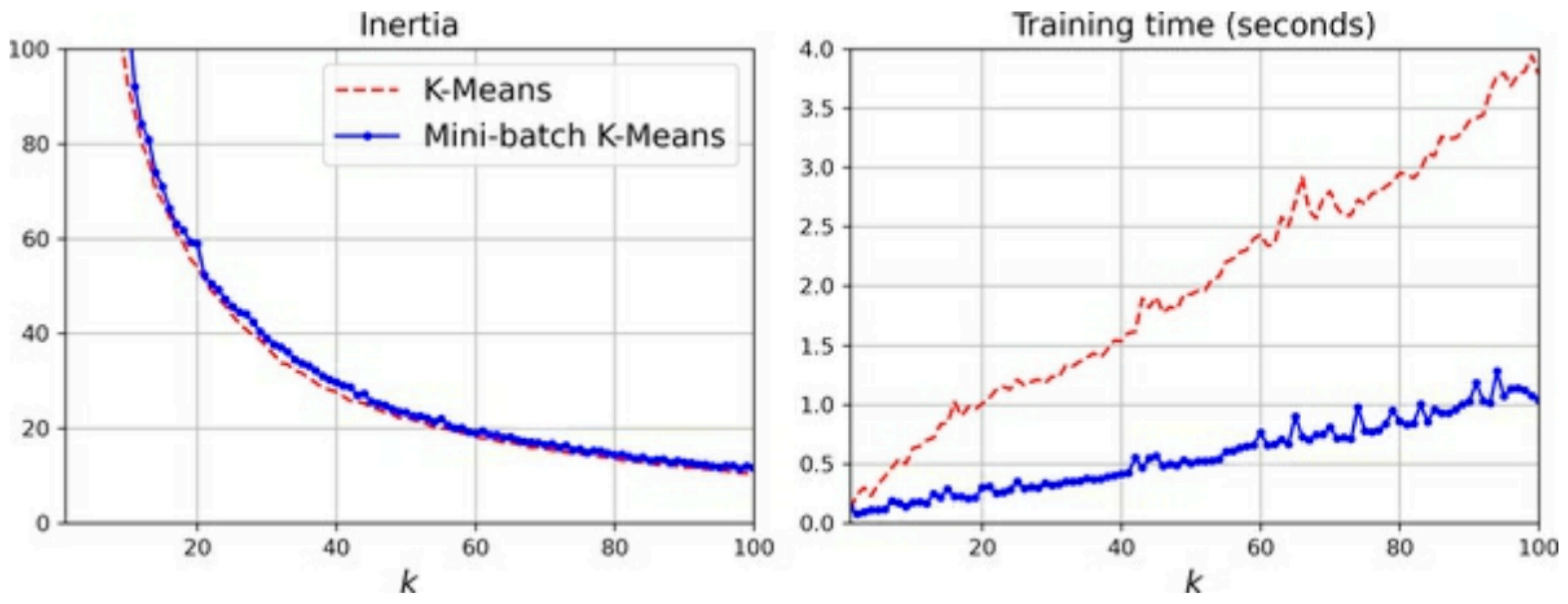
- An improvement proposed in 2006
- Chooses better starting centroids that are more distant from each other
- Much less likely to converge to a suboptimal solution
- The default method for the k-Means class we're using

Accelerated and Mini-Batch k-Means

- On large datasets with many clusters
- The calculation can be accelerated
- Using the triangle inequality to estimate distances using upper and lower bounds
- Doesn't always work, can make training slower

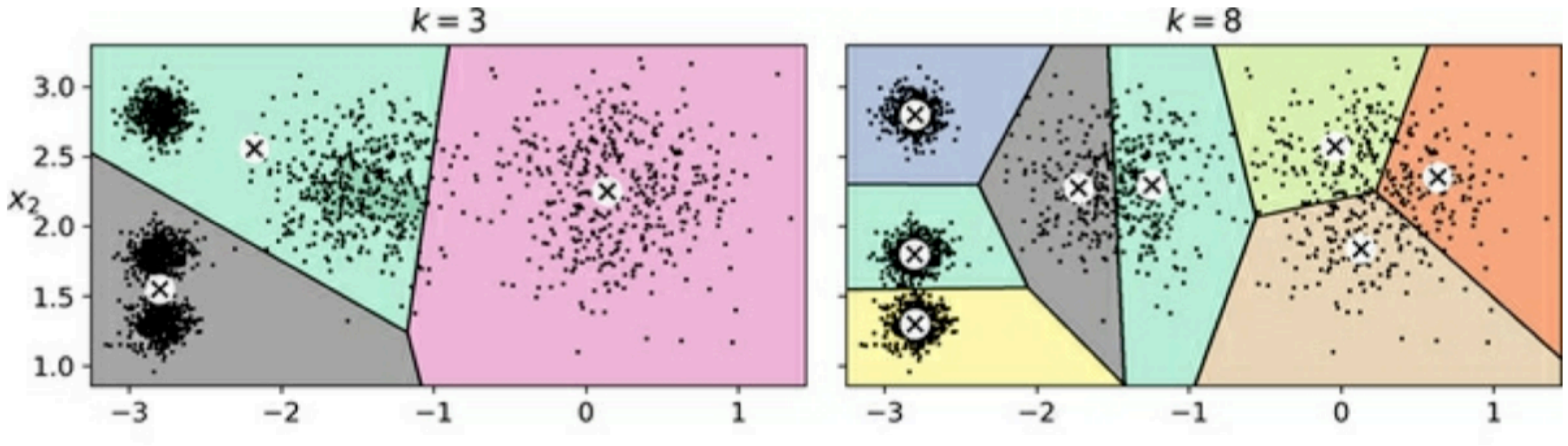
Mini-Batch k-Means

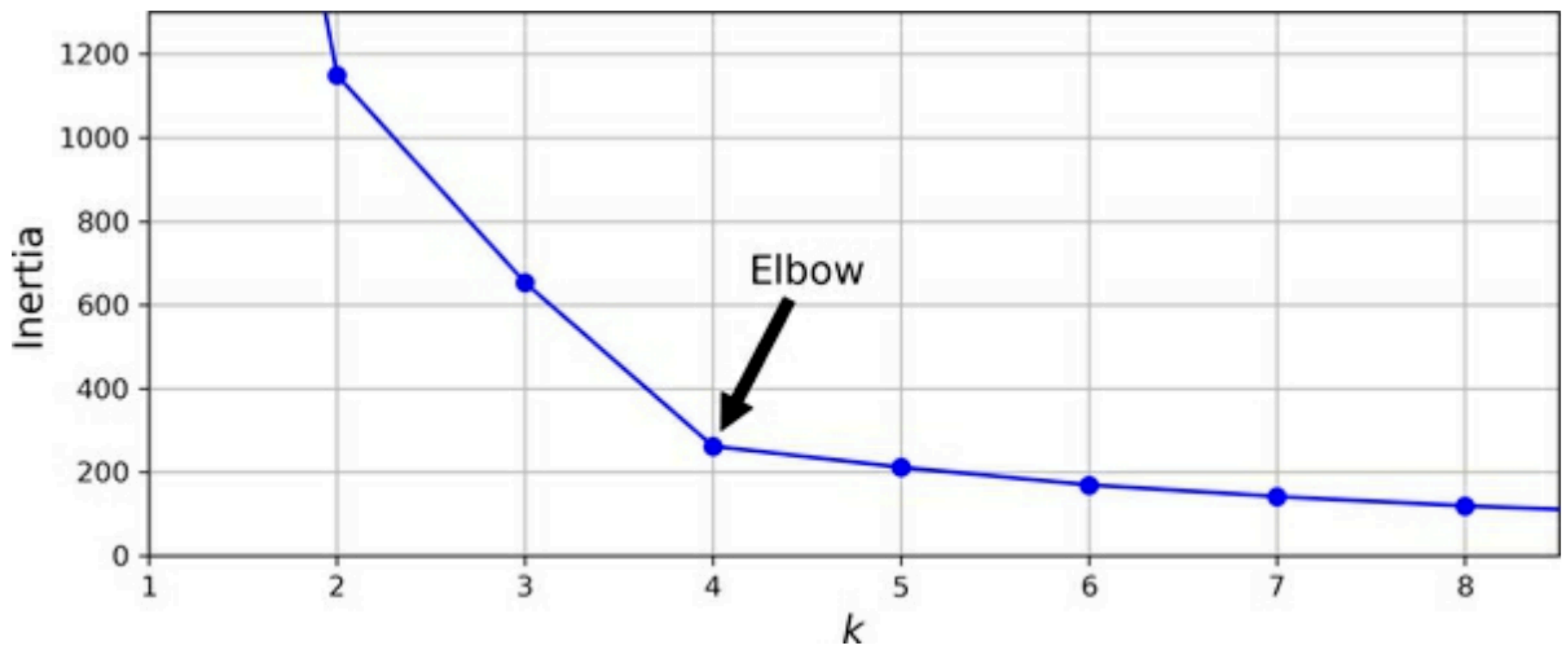
- Use only part of the data for each iteration
- Move centroids just slightly each time
- Speeds up algorithm
- Allows use of huge datasets



Choosing *k*

- More clusters means lower inertia
  - Even if it's overfitting the data



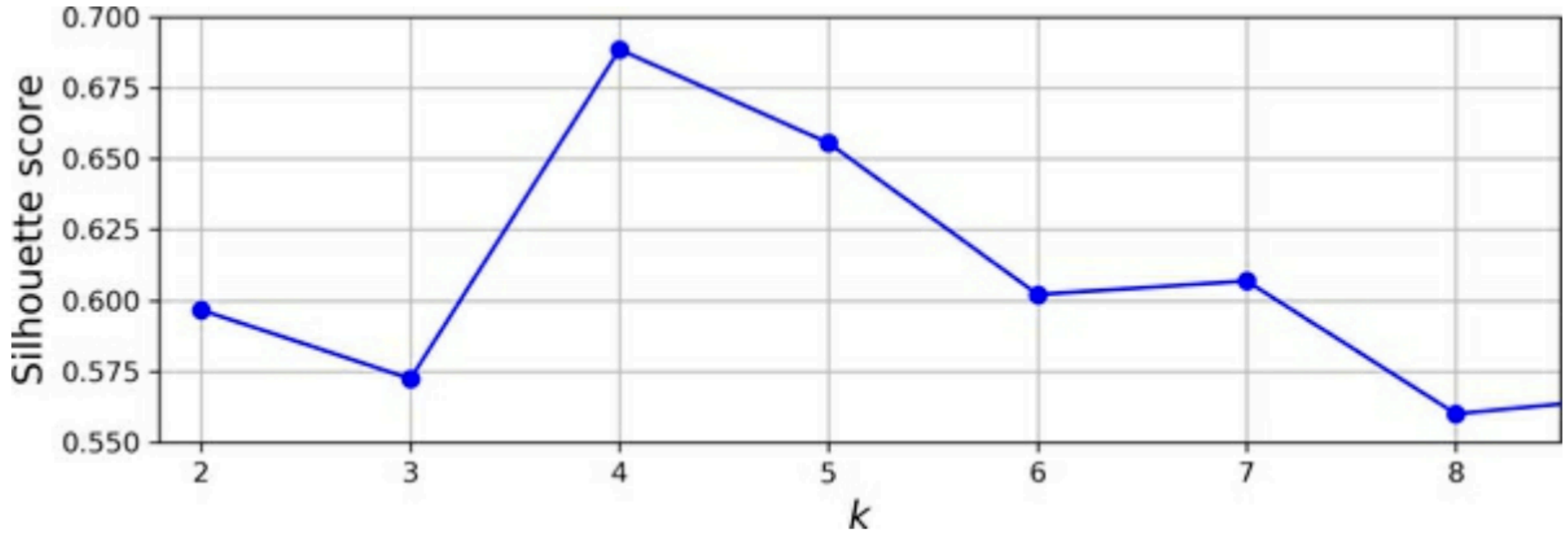


Silhouette Score

- Mean ***silhouette coefficient*** over all the instances
  - (***b*** - ***a***) / max(***a***, ***b***)
  - ***a*** is the mean distance to other instances in the same cluster
  - ***b*** is the mean distance to the instances in the next-closest cluster
- Varies from -1 to 1
  - +1 if an instance is well inside its own cluster and far from other clusters
  - -1 if an instance is assigned to the wrong cluster

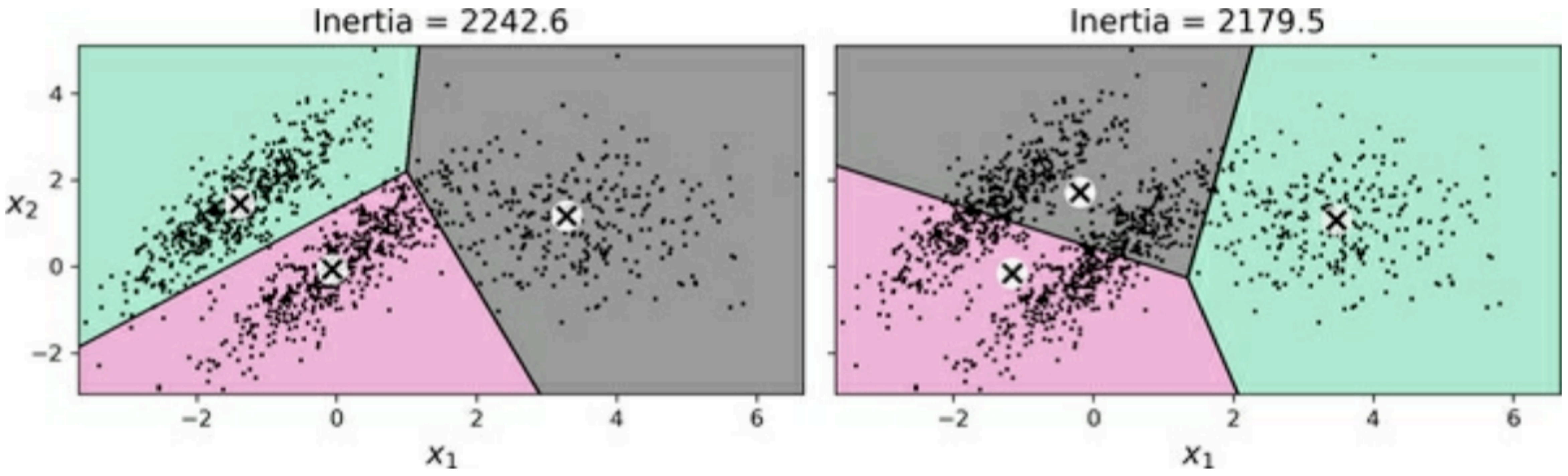
Results for 5-Cluster Data

- 4 is the best choice
- 5 is the second-best



Limits of k-Means

- Poor behavior when the clusters have
  - varying sizes,
  - different densities, or
  - nonspherical shapes
- Gaussian mixtures are better for these elliptical clusters



Using Clusters for Image Segmentation

- **Color segmentation**
  - Pixels with similar colors are assigned to the same segment
  - Useful for finding forest area in a landscape
- **Semantic segmentation**
  - Pixels that are part of the same object are assigned to the same segment
    - All pedestrians get sorted to the pedestrian segment
- **Instance segmentation**
  - Pixels that are part of the same individual; are assigned to the same segment
    - Each pedestrian is a different segment

Color Segmentation with k-Means

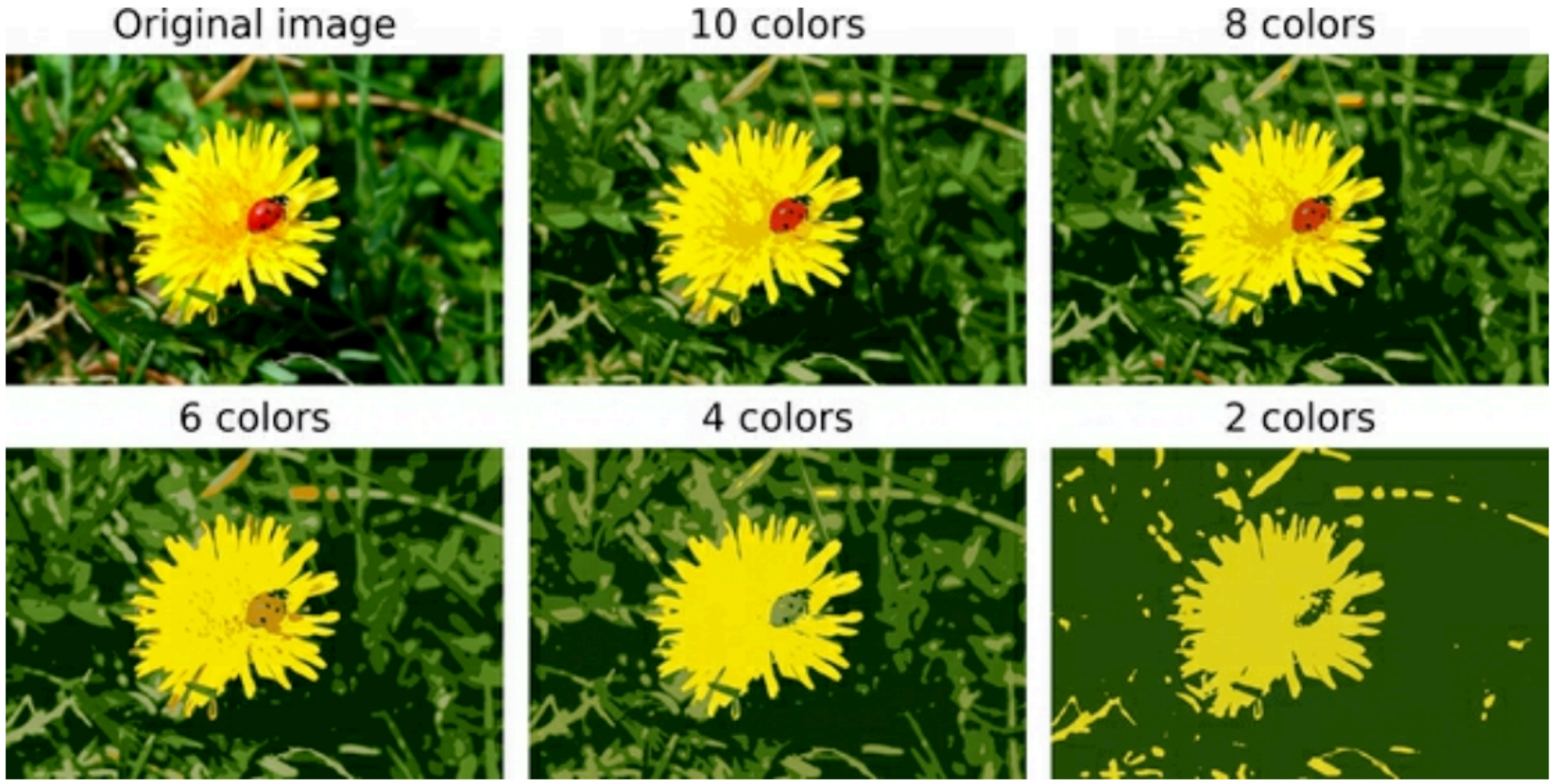
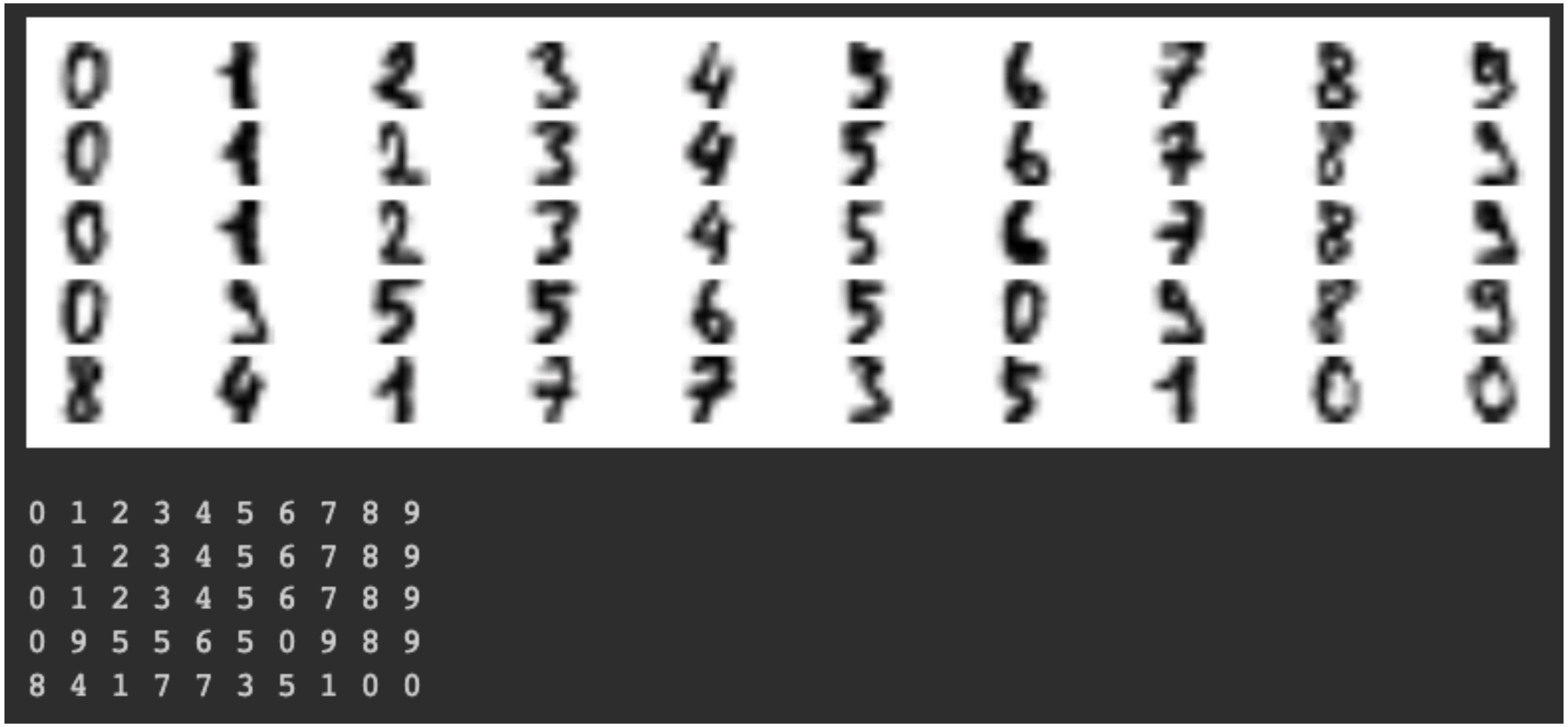


Figure 9-12. Image segmentation using k-means with various numbers of color clusters

Using Clustering for Semi-Supervised Learning

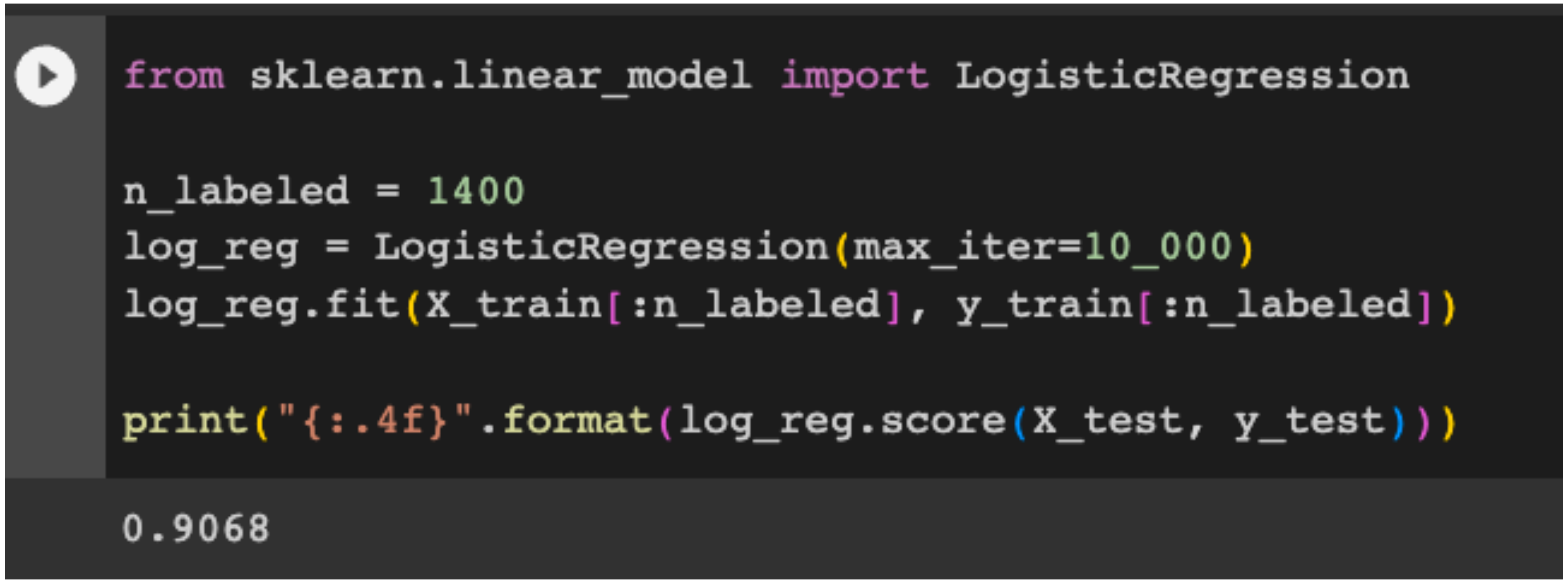
- Many unlabeled instances, and a few labeled instances
- This dataset has 1,797 images with labels



```
0 1 2 3 4 5 6 7 8 9
0 1 2 3 4 5 6 7 8 9
0 1 2 3 4 5 6 7 8 9
0 9 5 5 6 5 0 9 8 9
8 4 1 7 7 3 5 1 0 0
```

Fitting a Model

- Fit using all images and labels
- Over 90% accuracy

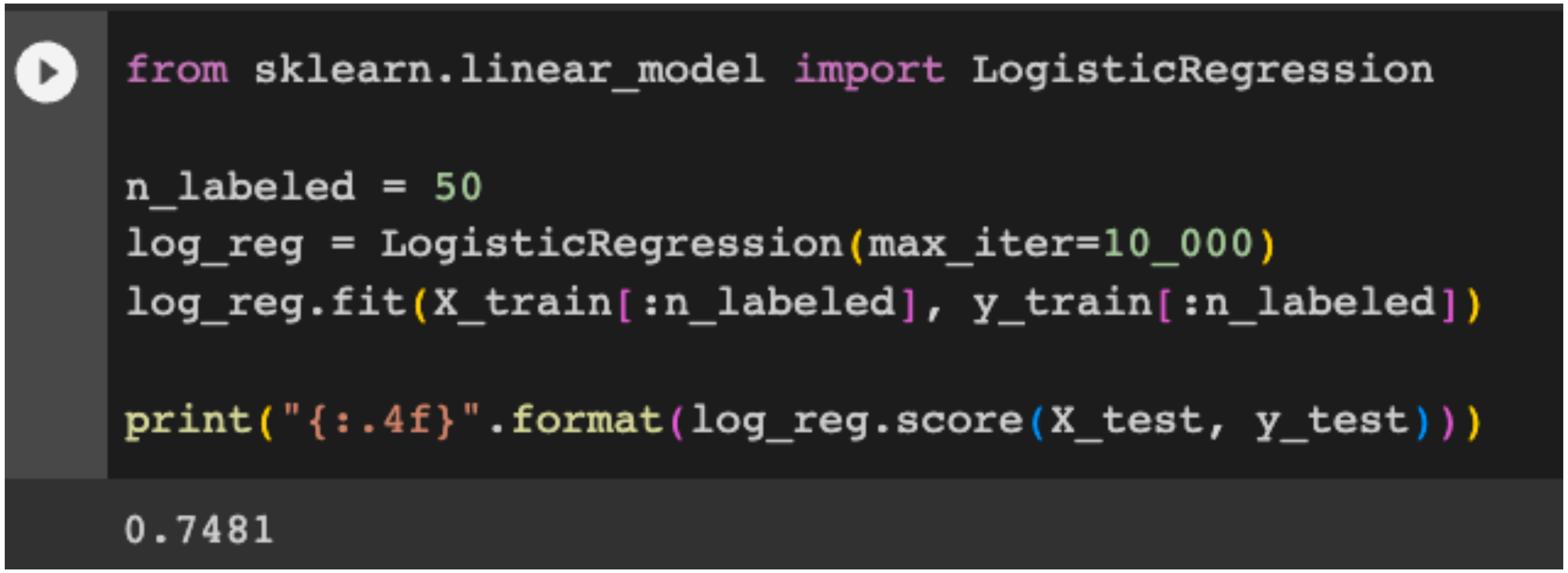


```python
from sklearn.linear_model import LogisticRegression

n_labeled = 1400
log_reg = LogisticRegression(max_iter=10_000)
log_reg.fit(X_train[:n_labeled], y_train[:n_labeled])

print("{:.4f}".format(log_reg.score(X_test, y_test)))
```

```
0.9068
```

Fitting a Model to 50 Instances

- Only about 75% accurate

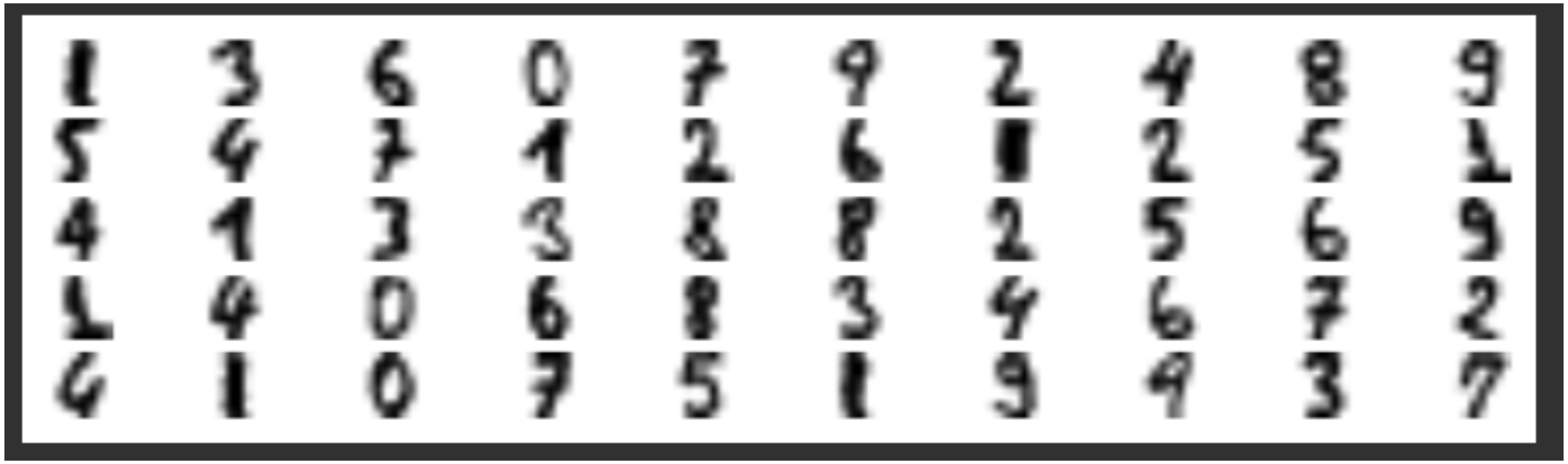
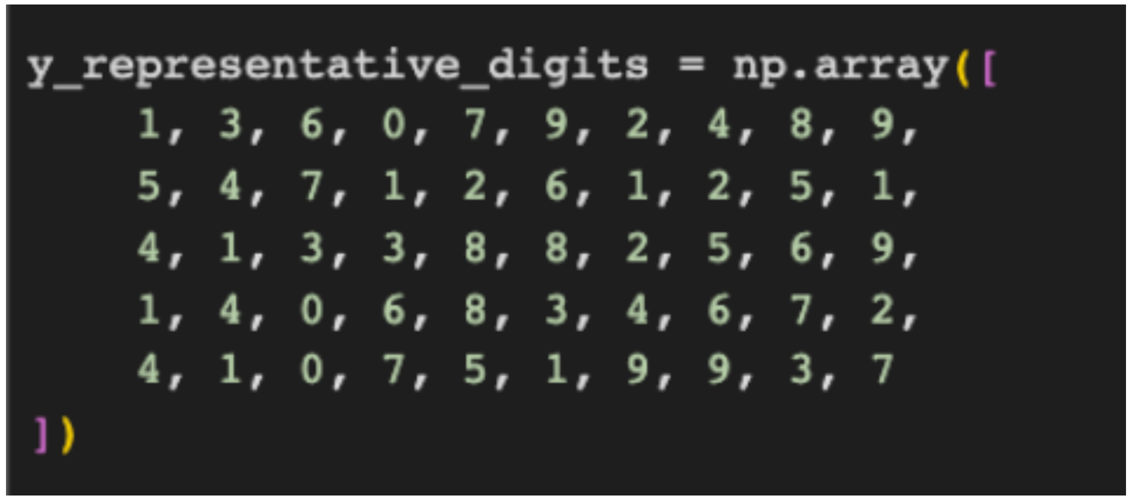


```
from sklearn.linear_model import LogisticRegression

n_labeled = 50
log_reg = LogisticRegression(max_iter=10_000)
log_reg.fit(X_train[:n_labeled], y_train[:n_labeled])

print("{:.4f}".format(log_reg.score(X_test, y_test)))
```

```
0.7481
```

Using a k-Means Model to form 50 Clusters

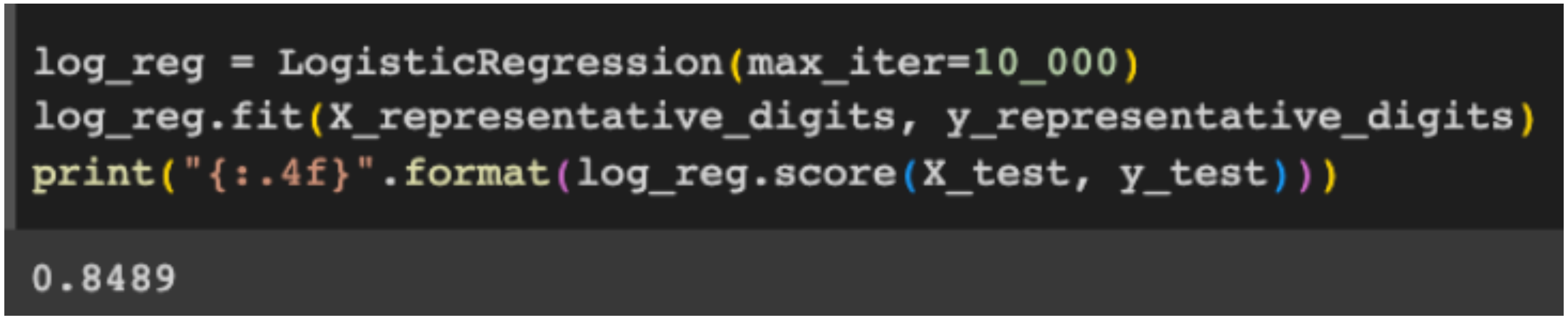
- Find representative images nearest to the centroids
- These are better images to use for training
- Label them by hand

```
y_representative_digits = np.array([
    1, 3, 6, 0, 7, 9, 2, 4, 8, 9,
    5, 4, 7, 1, 2, 6, 1, 2, 5, 1,
    4, 1, 3, 3, 8, 8, 2, 5, 6, 9,
    1, 4, 0, 6, 8, 3, 4, 6, 7, 2,
    4, 1, 0, 7, 5, 1, 9, 9, 3, 7
])
```

Training from Representative Images

- Accuracy increases to about 85%

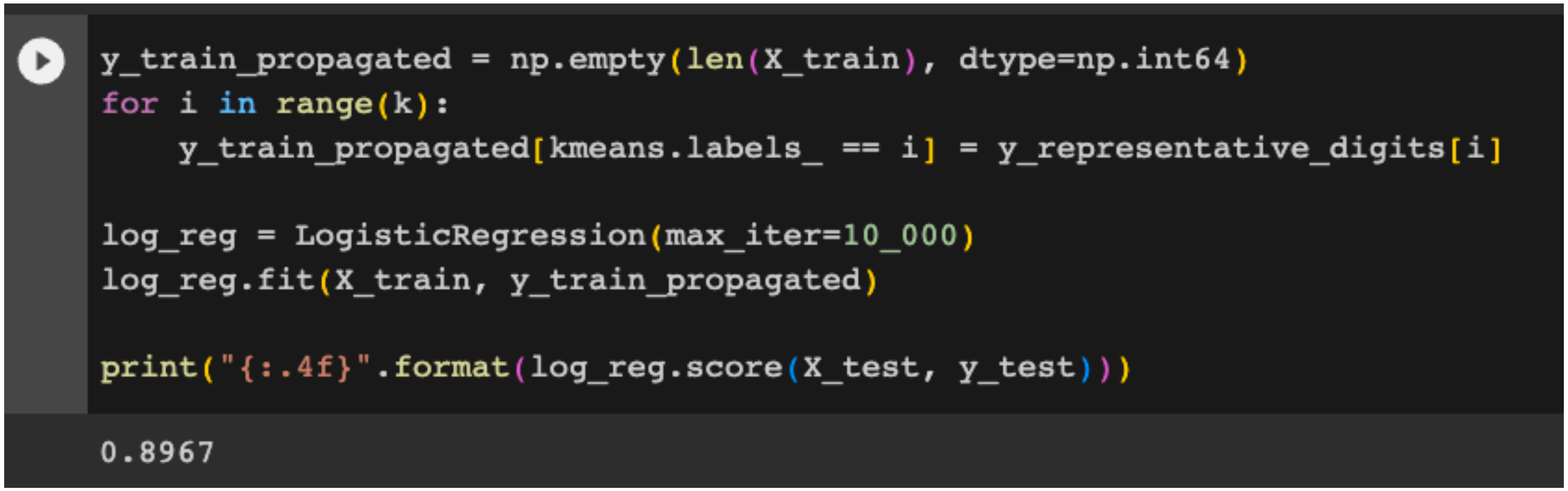


```
log_reg = LogisticRegression(max_iter=10_000)
log_reg.fit(X_representative_digits, y_representative_digits)
print("{:.4f}".format(log_reg.score(X_test, y_test)))
```

```
0.8489
```

Propagating Labels to All Instances

- In each of the 50 clusters
- Accuracy increases to 89%

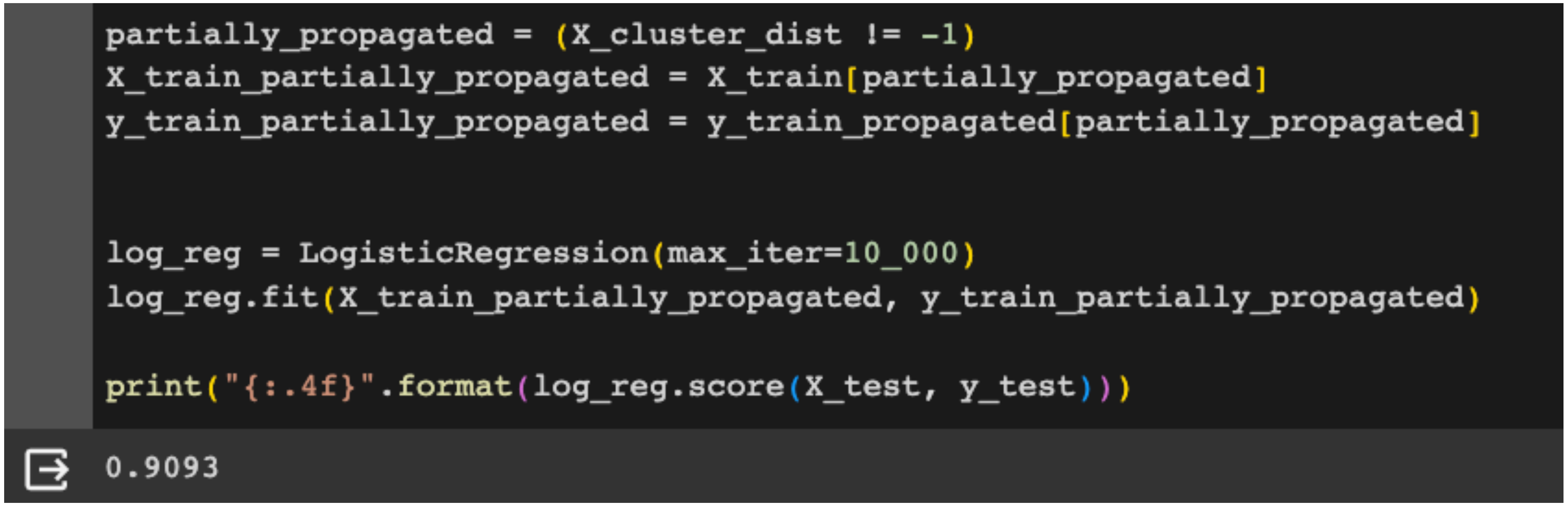


```python
y_train_propagated = np.empty(len(X_train), dtype=np.int64)
for i in range(k):
    y_train_propagated[kmeans.labels_ == i] = y_representative_digits[i]

log_reg = LogisticRegression(max_iter=10_000)
log_reg.fit(X_train, y_train_propagated)

print("{:.4f}".format(log_reg.score(X_test, y_test)))
```

```
0.8967
```

Removing Outliers

- Remove the 1% of images furthest from centroids
- Accuracy increases to over 90%



```
partially_propagated = (X_cluster_dist != -1)
X_train_partially_propagated = X_train[partially_propagated]
y_train_partially_propagated = y_train_propagated[partially_propagated]


log_reg = LogisticRegression(max_iter=10_000)
log_reg.fit(X_train_partially_propagated, y_train_partially_propagated)

print("{:.4f}".format(log_reg.score(X_test, y_test)))
```

```
0.9093
```

Summary

• Train on all data with labels	91%
• Train on first 50 images	75%
• Train on 50 representative images	85%
• Propagate labels to all images	89%
• Remove outliers	91%

Active Learning

- A human expert interacts with the learning algorithm
- ***Uncertainty sampling***
  - The model is trained on the instances labeled so far
  - Makes predictions on the unlabeled instances
  - Instances are given to the expert for labeling, those
    - With the most uncertainty, or
    - That would result in the largest model change, or
    - That different models disagree on
  - Iterate this process until the performance improvement stops being worth the labeling effort

Kahoot!

Ch 9a

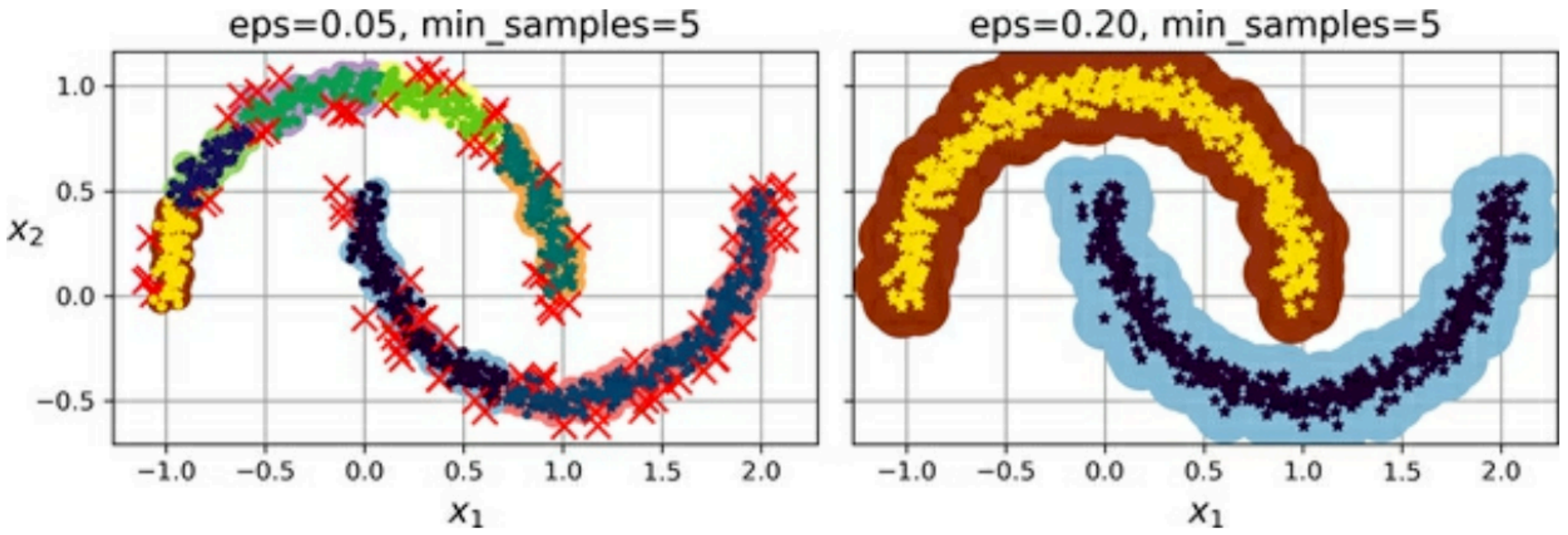
DBSCAN

DBSCAN

- Density-Based Spatial Clustering of Applications with Noise
- Clusters are continuous regions of high density
    - For each instance, count how many instances are near it
        - Distance <***Ɛ*** (***Ɛ***-neightorhood)
    - If an instance has at least **min_samples** in its ***Ɛ***-neightorhood
        - It's a ***core instance*** -- located in a dense region
    - All instances in the neighborhood of a core instance belong to the same cluster
    - An instance that is not a core instance and doesn't have one in its neighborhood is an anomaly

DBSCAN

- If $\mathcal{E}$ is too small, clusters are broken up, on the left below
- On the right, a correct value of $\mathcal{E}$



Other Clustering Algorithms

- **Agglomerative clustering**
  - Connects clusters together with each iteration like bubbles
- **BIRCH**
  - Balanced Iterative Reducing and Clustering using Hierarchies
  - Designed for very large datasets
  - Can be faster than batch k-Means as long as there are less than 20 features
  - During training, builds a tree structure with just enough information to assign new instances to clusters

Other Clustering Algorithms

- **Mean-shift**
  - Starts with a circle centered on each instance
  - Shifts the circle to center on the mean of clusters inside
  - Iterates until the circles stop moving
  - Combines close circles to form clusters
  - Can find clusters of any shape
  - Only one hyperparameter: ***bandwidth*** (the circle size)

Other Clustering Algorithms

- **Affinity propagation**
  - Instances exchange messages and elect ***exemplars***
  - Each exemplar and the instances that selected it are the clusters
  - Can find clusters of different sizes
  - Complexity of order ***m*²** so not good for large datasets

Other Clustering Algorithms

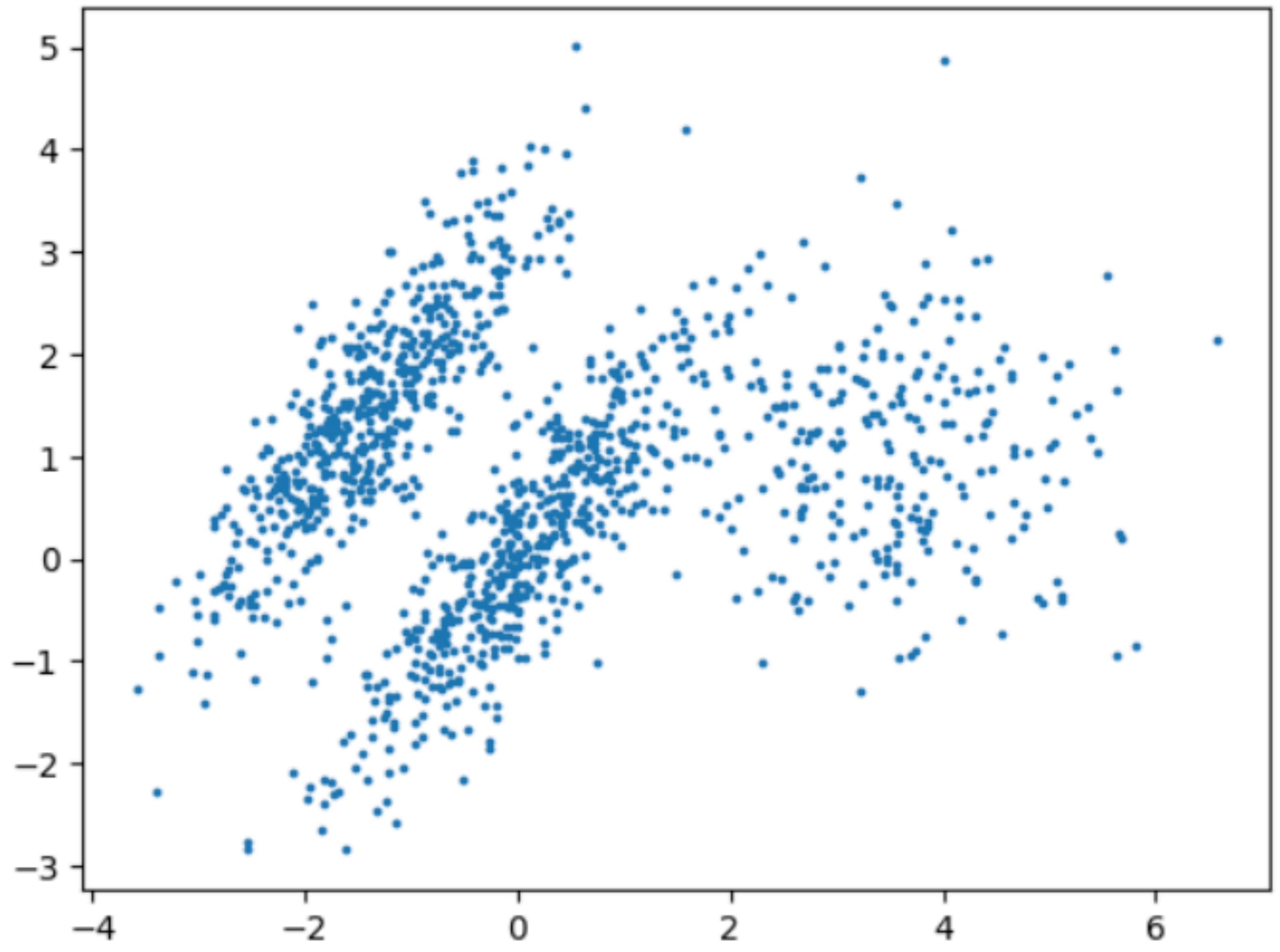
- **Spectral clustering**
    - Makes a similarity matrix between clusters
    - Reduces dimensionality
    - Uses another clustering algorithm like k-means in this low dimensional space
    - Does not scale well to large numbers of instances
    - Does not handle different cluster sizes well

Gaussian Mixtures

Gaussian Mixtures

- Assume instances are a group of Gaussian distributions
- Input the number of clusters
- Estimate Gaussian parameters

```
from sklearn.datasets import make_blobs
import numpy as np
import matplotlib.pyplot as plt

X1, y1 = make_blobs(n_samples=1000, centers=((4, -4), (0, 0)), random_state=42)
X1 = X1.dot(np.array([[0.374, 0.95], [0.732, 0.598]]))
X2, y2 = make_blobs(n_samples=250, centers=1, random_state=42)
X2 = X2 + [6, -8]
X = np.r_[X1, X2]
y = np.r_[y1, y2]

plt.scatter(X[:, 0], X[:, 1], s=3)
plt.show()
```

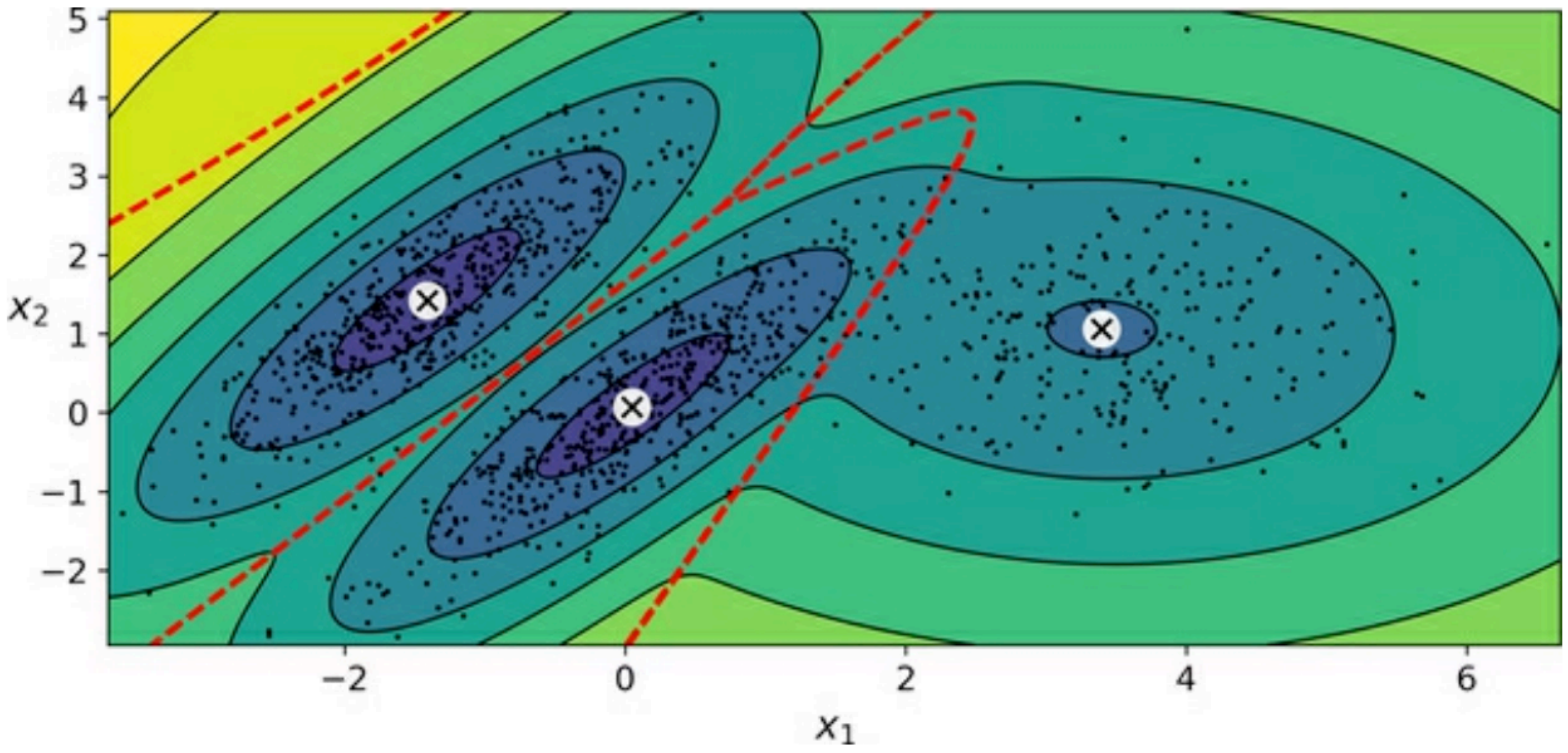


Expectation-Maximization (EM)

- Initialize cluster parameters randomly
  - Assign instances to clusters (expectation)
  - Update clusters (maximization)
- A generalization of k-means
  - Allowing various cluster size, shape, and orientation
- Like k-means, it can converge to poor solutions
- Repeat **n_init** times (default 1, set to 10)

Gaussian Mixture Model

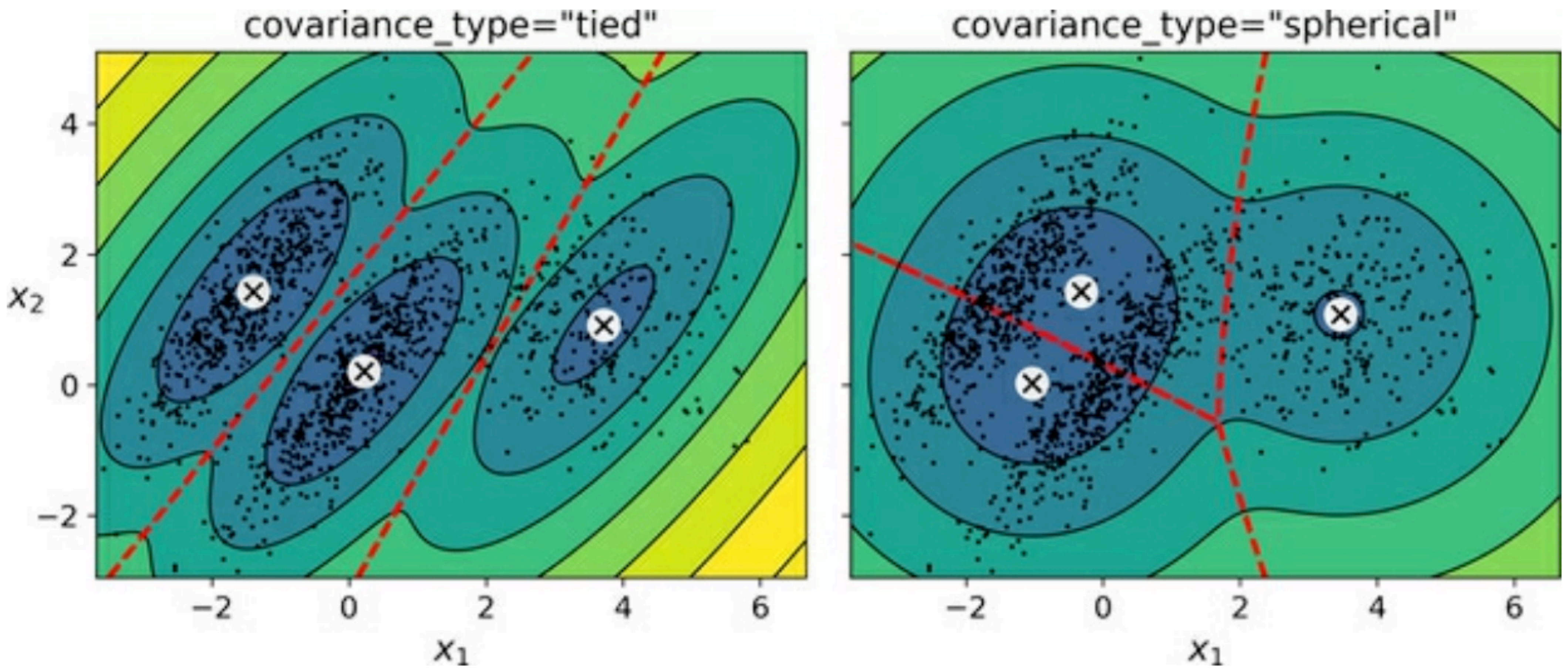
- Fits this simple data well



Limiting Shapes

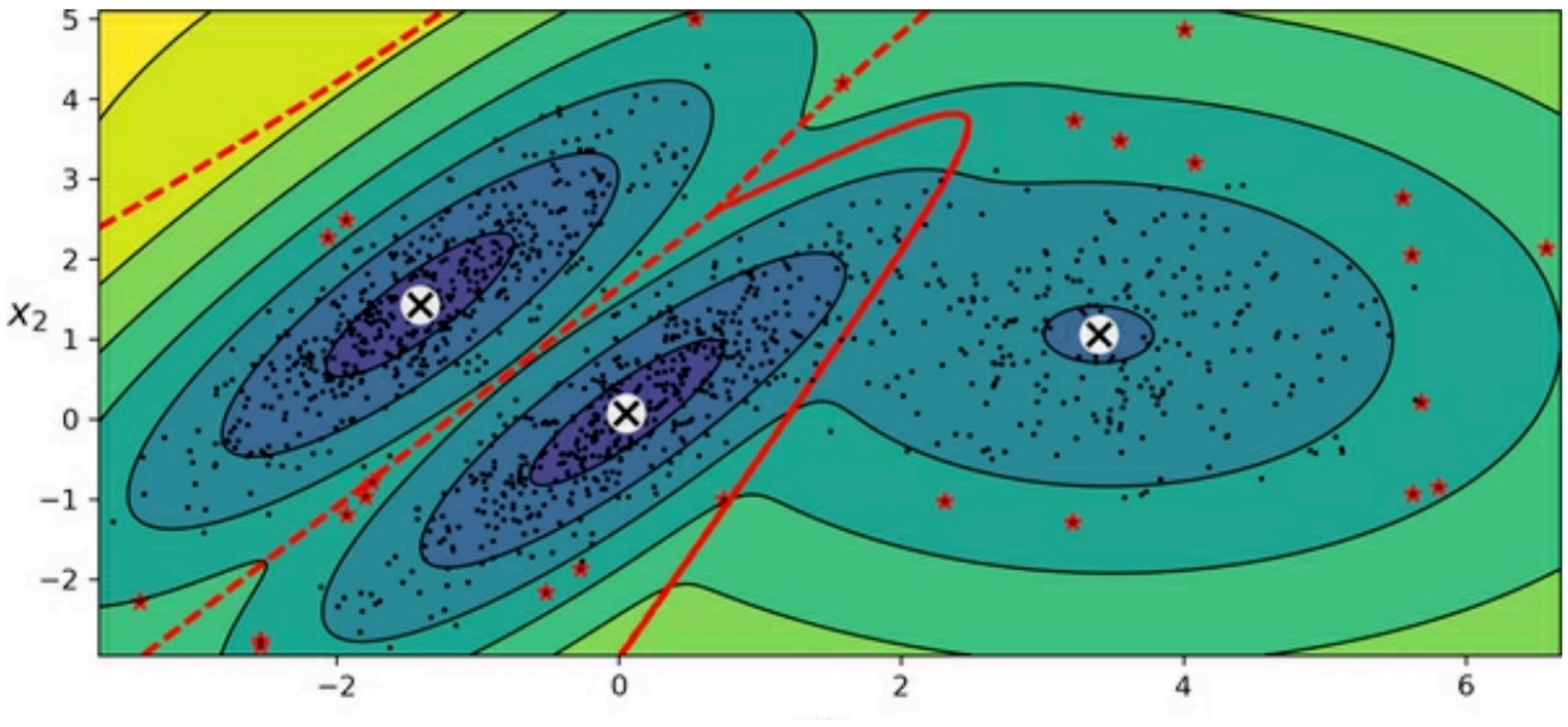
- **covariance_type** hyperparameter
  - "**spherical"**
    - but can have different sizes
  - "**diag"**
    - can be ellipsoids but axes must be aligned with coordinate axes
  - "**tied"**
    - all ellipsoids must have the same shape, size, and orientation
  - "**full"**
    - no constraints

Constrained Models



Anomaly Detection

- An instance in a low-density region is an anomaly
- Choose a threshold
  - Ex: 2% of products are defective



Selecting the Number of Clusters

- Minimize **Theoretical Information Criterion** such as
  - **Bayesian Information Criterion (BIC)**
  - **Akaike Information Criterion (AIC)**

$$BIC = \log(m)\,p - 2\log\left(\widehat{\mathscr{L}}\right)$$

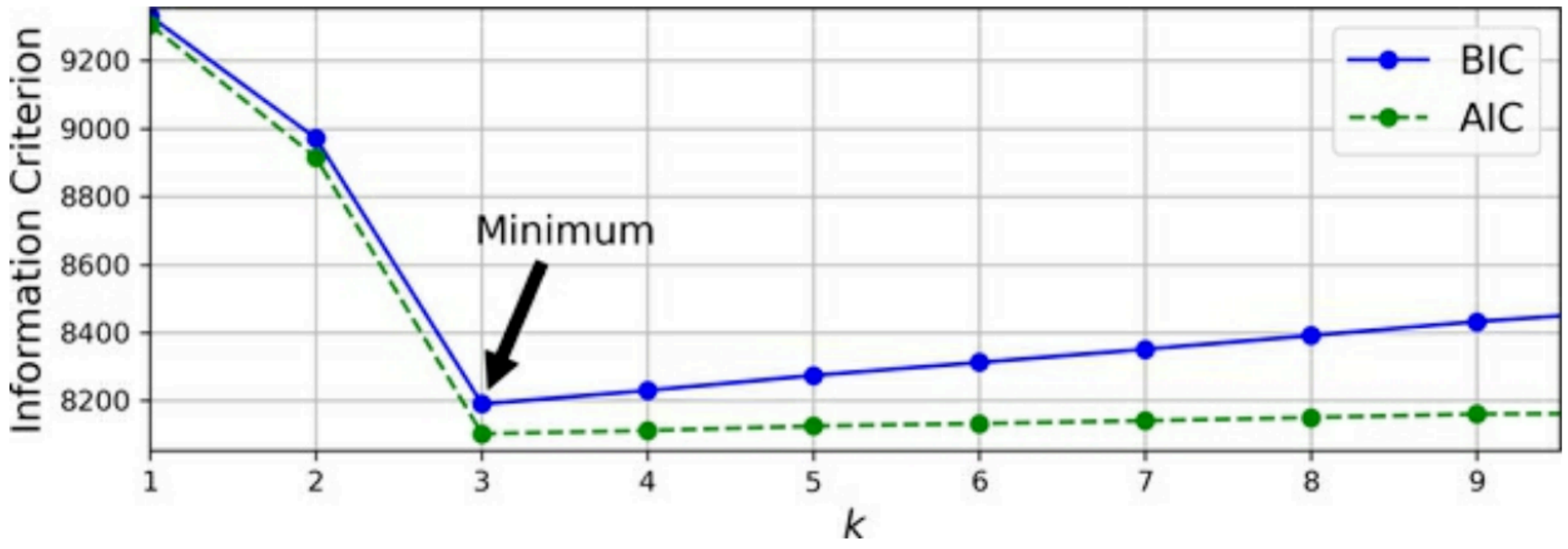
$$AIC = 2p - 2\log\left(\widehat{\mathscr{L}}\right)$$

In these equations:

- ***m*** **is the number of instances, as always.**
- ***p*** **is the number of parameters learned by the model.**
- $\widehat{\mathscr{L}}$ **is the maximized value of the *likelihood function* of the model.**

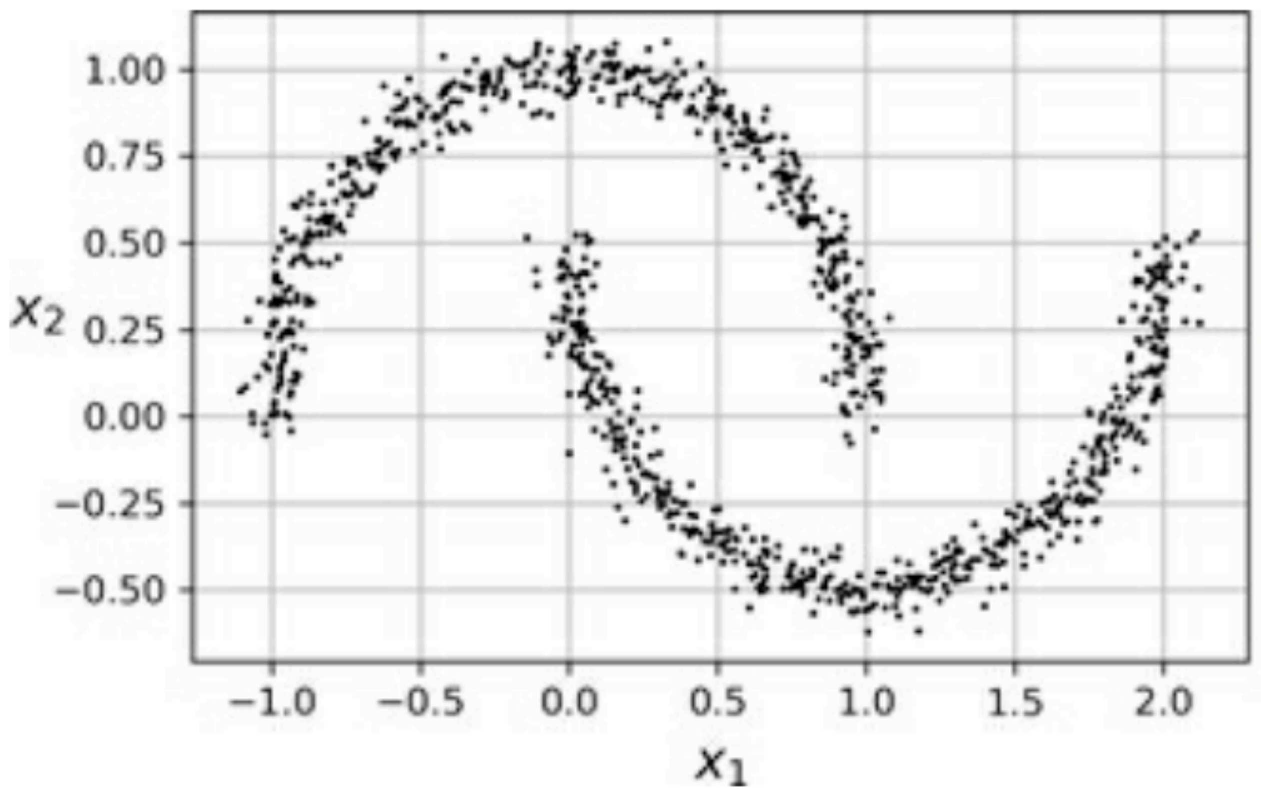
BIC and AIC

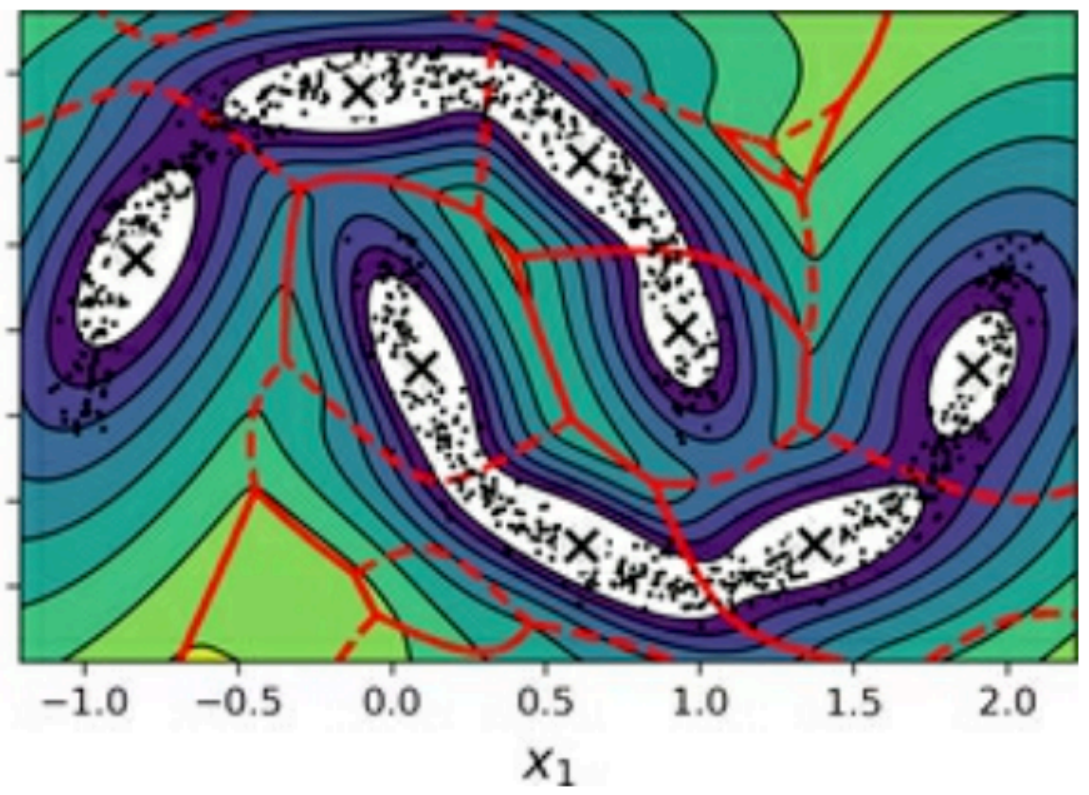
- Both measures are minimized for k=3



Bayesian Gaussian Mixture Models

- Set **n_components** to a large value, more than the number of clusters you expect
- Algorithm will eliminate unnecessary clusters automatically
- Works for the simple 3 Gaussian cluster data
- Not so well for moons





Other Algorithms

- **Fast-MCD (Minimum Covariance Determinant)**
  - Useful for outlier detection, to clean up a dataset
  - Assumes that inliners are from a single Gaussian distribution
  - Classified the others as **outliers**

Other Algorithms

- **Isolation Forest**
  - Efficient for outlier detection, especially in high-dimensional datasets
  - Builds a random forest, randomly picking features and thresholds at each node
  - Dataset gets chopped into pieces until each instance is alone
  - Anomalies get isolated in fewer steps
- **Local Outlier Factor (LOF)**
  - Compares density of instances around a given instance to the density around its neighbors
  - Anomalies are more isolated than the neighbors

Other Algorithms

- **One-class SVM**
  - Suited for novelty detection
  - Maps instances to a high-dimensional space
  - Uses a linear SVM classifier to separate the instances from the origin
  - Corresponds to finding a small region that encompasses all the instances in the original space
  - A new instance that is outside that region is an anomaly
- **PCA and other dimensionality reduction techniques**
  - Reconstruction error is larger for anomalies

Kahoot!

Ch 9b