BTS | Barcelona Technology School

Classical Data Analysis

Master in Big Data Solutions 2020-2021

Filipa Peleja

Víctor Pajuelo

filipa.peleja@bts.tech

victor.pajuelo@bts.tech

Today's class

Contents

- Unsupervised Learning
  - Look at the beauty of using less data
  - Look at clustering techniques such as KMeans
  - Understand the limits of KMeans
  - Look at the usages of KMeans

Today's objective

- Being able to use unsupervised learning for clustering datasets, segmenting images and doing semi-supervised classification

Let's git things done!

Let's see it again

Pull Session 9 notebooks

```
$ git clone https://github.com/vfp1/bts-cda-2020.git

# If you have done that already
$ git pull origin master
```

BTS | Barcelona Technology School

MBDS 2020-2021

Unsupervised Learning

Unsupervised Learning

The next AI revolution

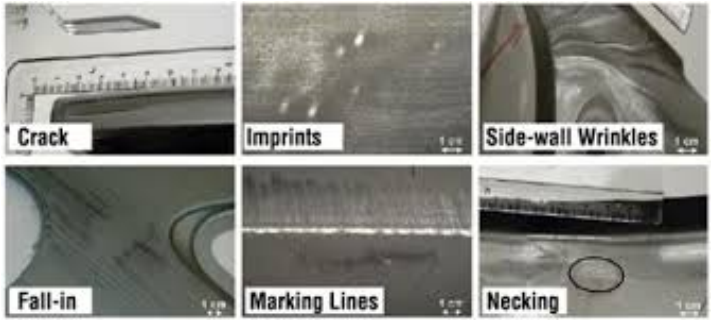
- The vast majority of the available data is unlabeled:
  - We have input features **X**, but we do not have the labels **y**
- Most of the investment on research is going to supervised learning, because most of the applications are there
- But the **real future** is really within the unsupervised realm, which is closer to how we humans learn
  - Actually is closer to active/reinforcement learning, which are a type of subsets of semi-unsupervised learning

Unsupervised Learning

The Data Problem

- Let's say that a company calls you to implement AI in their domain. For instance, checking metal stress in their mass production facilities
- You install cameras in their facility and start collecting pictures. You build up an amazingly large dataset in few weeks. But you've got no labels…
- Now you need a team of trained "metal experts" to manually label all the pictures, which is insanely expensive and time consuming
- Every time, the company changes their metal suppliers, we will need to relabel and retrain…

Wouldn't it be great if we could have an **algorithm to exploit unlabeled data without needing humans to label each picture?** Welcome to the **world of unsupervised learning**.



Unsupervised Learning

The cake

*"If intelligence was a cake, **unsupervised learning** would be the cake, **supervised learning** would be the icing on the cake, and **reinforcement learning** would be the cherry on the cake"*

Yan LeCun, AI guru and master baker.

In other words, there is an enormous potential in unsupervised learning for which we barely started sinking our teeth (and our spoons) into.

Unsupervised Learning

The cake

- "Pure" Reinforcement Learning (cherry)
  - The machine predicts a scalar reward given once in a while.
  - **A few bits for some samples**
- Supervised Learning (icing)
  - The machine predicts a category or a few numbers for each input
  - Predicting human-supplied data
  - **10→10,000 bits per sample**
- Unsupervised/Predictive Learning (cake)
  - The machine predicts any part of its input for any observed part.
  - Predicts future frames in videos
  - **Millions of bits per sample**

- (Yes, I know, this picture is slightly offensive to RL folks. But I'll make it up)

Unsupervised Learning

Main tasks

- Without maybe being too aware of it, we have already performed some unsupervised learning tasks, i.e.
  - Dimensionality Reduction
- However, there are **more tasks involved**:

Task	Objective	Applications
Clustering	Group similar instances together into *clusters*	Data analysis, customer segmentation, recommender systems, search engines, image segmentation, semi-supervised learning, dim reduction, etc.
Anomaly detection	Learn what “normal” data looks like, then use that to detect abnormal instances	Detection of defective items in product lines, new trend in time series, etc.
Density estimation	Estimating the Probability Density Function (PDF) of the random process that generated the dataset.	Anomaly detection, data analysis and visualization

BTS | Barcelona Technology School

MBDS 2020-2021

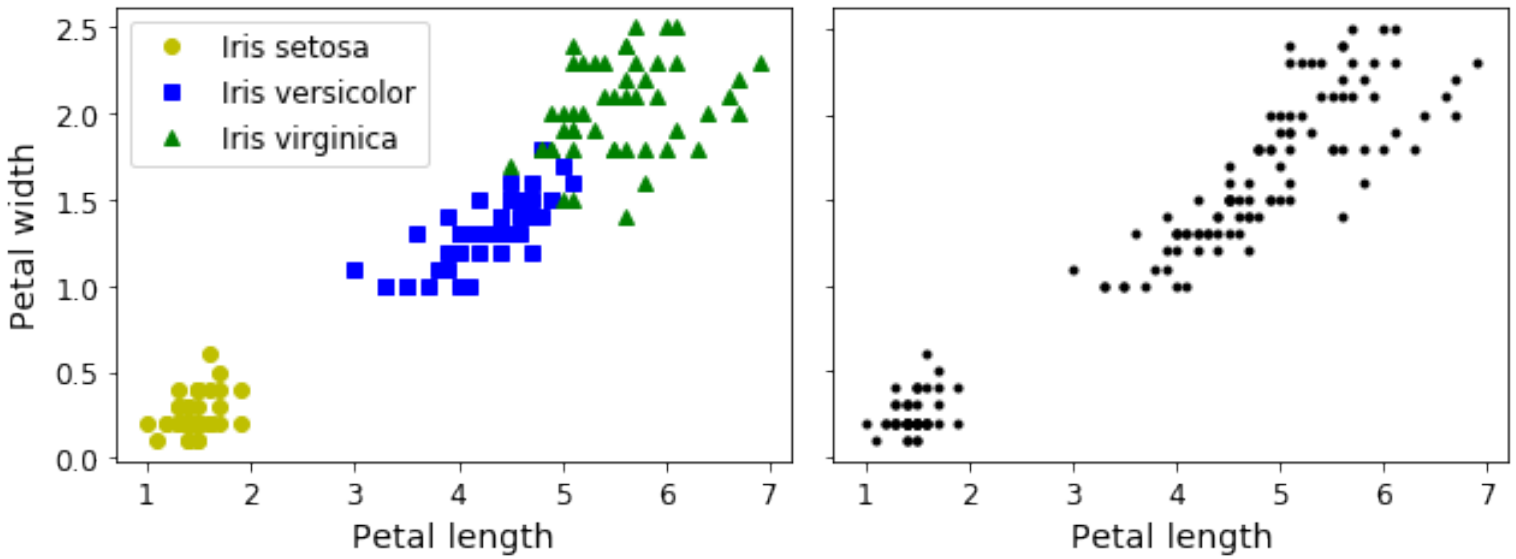
Unsupervised Learning

Clustering

Clustering

Introduction

- If we go to the mountains and look at different plants, we will **need an expert to tell us exactly what species are** we looking at, but we will not need anyone to **identify groups of similarly looking objects**
- This is called **CLUSTERING**

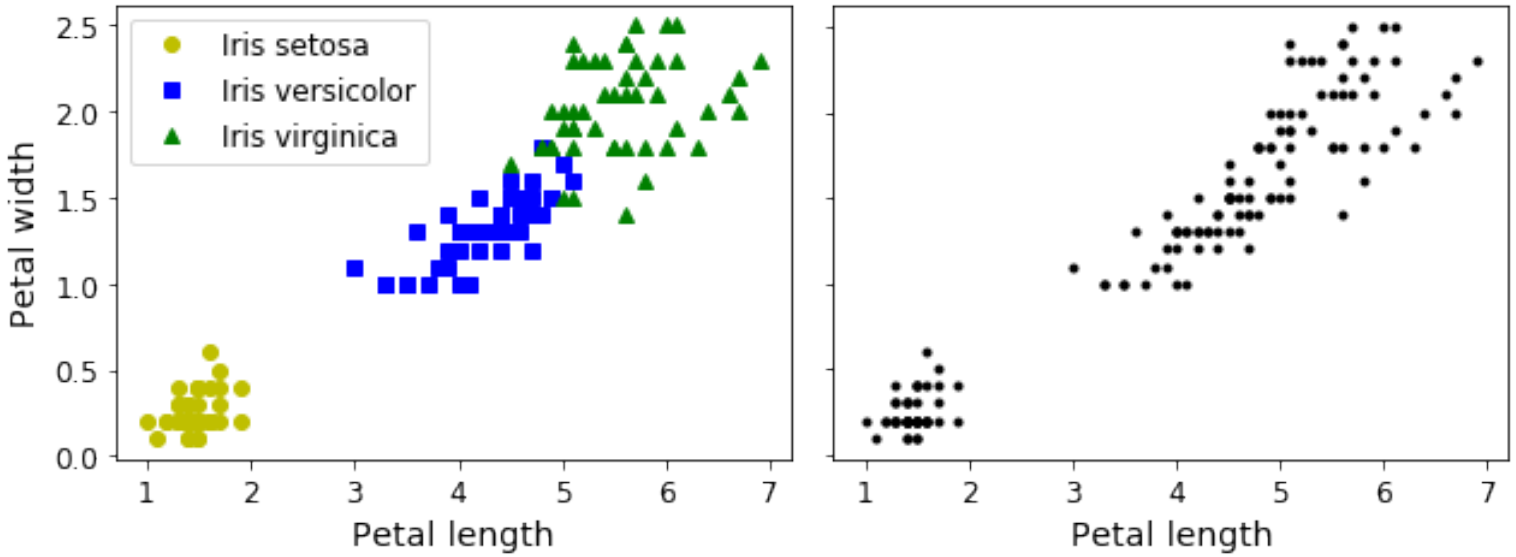


*Code to reproduce the graphs can be seen at **UUID - #S9C1***

Clustering

Introduction

- Just like classification, each instance gets assigned to a group
- Unlike classification, clustering is fully unsupervised
- The **left graph** represents a **labelled dataset**, for which classification algorithms such as Logistic Regression, SVMs, Random Forests are very well suited
- On the **right side** the same dataset is represented, but **without labels**, so a classification algorithm cannot be used anymore
- So, *how do we address this type of datasets?*

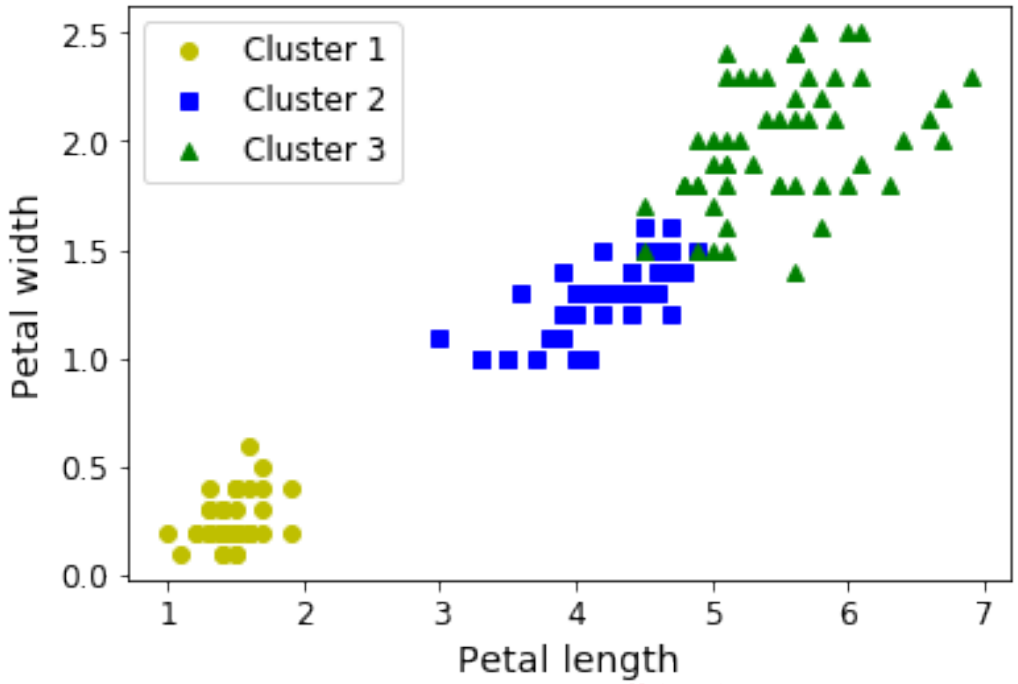


*Code to reproduce the graphs can be seen at **UUID - #S9C1***

Clustering

Introduction

- Using clustering algorithms, we can separate the three classes
- Actually GMMs are quite good with this datasets, since it only makes a mistake on 5 out of the 150 instances, i.e. reaching a 3% of error



*Code to reproduce the graphs can be seen at **UUID - #S9C1***

Clustering

What is a cluster?

- There is really not an universal definition of what constitutes a cluster
- In general terms is an aggrupation of instances that share certain similarities according to the rules imposed by an algorithm
- It all depends on the context, since different algorithms will capture different types of clusters
  - Some algorithms look for instances centered around a point (centroids)
  - Some others look for regions of densely packed instances
  - Some algorithms are hierarchical, looking for clusters of clusters
  - Etc.

BTS | Barcelona Technology School

MBDS 2020-2021

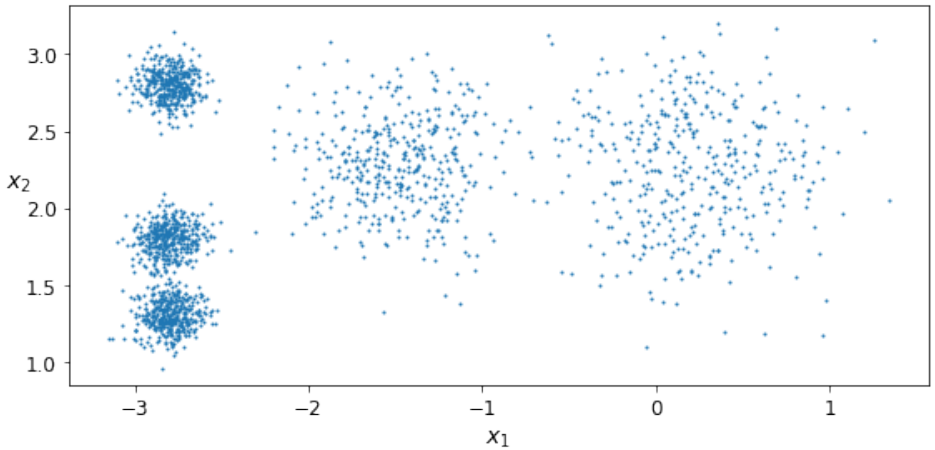
Unsupervised Learning

K-Means

K-Means

Intro

- The K-Means is a simple algorithm that can cluster such blob data in a very quickly and efficient manner, usually in few iterations
- It was created at the same time by Stuart Lloyd (working at Bell Labs) in 1957 and by Edward Forgy in 1965
- So you will often see it referenced in the literature as the Lloyd-Forgy algorithm



*Code to reproduce the graphs can be seen at **UUID - #S9C2***

K-Means

Usage in Sklearn

- The usage of K-Means in Sklearn is extremely simple
- We will cluster the blobs above by using K-Means
- We basically call `KMeans` class and assign our desired number of clusters
- In this case is kind of clear the we need to use 5 clusters, but usually this is not so straightforward

```
1 from sklearn.cluster import KMeans
```

```
1 k = 5
2 kmeans = KMeans(n_clusters=k, random_state=42)
3 y_pred = kmeans.fit_predict(X)
```

*Code to reproduce the graphs can be seen at **UUID - #S9C3***

K-Means

Labels and Cluster Centers

- Each instance was assigned to one of the five clusters
- Remember that this is unsupervised learning, the "labels" are assigned by the algorithm
  - Those should not be confused by the class labels in classification
- We can always access the assigned predicted labels through the `labels_` method, as well as the centroids through the `clusters_centers_` methods

```
1 kmeans.labels_
```

```
array([4, 1, 0, ..., 3, 0, 1], dtype=int32)
```

```
1 kmeans.cluster_centers_
```

```
array([[ 0.20876306,  2.25551336],
       [-2.80389616,  1.80117999],
       [-1.46679593,  2.28585348],
       [-2.79290307,  2.79641063],
       [-2.80037642,  1.30082566]])
```

*Code to reproduce the graphs can be seen at **UUID - #S9C3***

K-Means

Prediction

- We can easily predict new instances that we add by passing a predict method as in any supervised learning method

```
1 X_new = np.array([[0, 2], [3, 2], [-3, 3], [-3, 2.5]])
2 kmeans.predict(X_new)
```

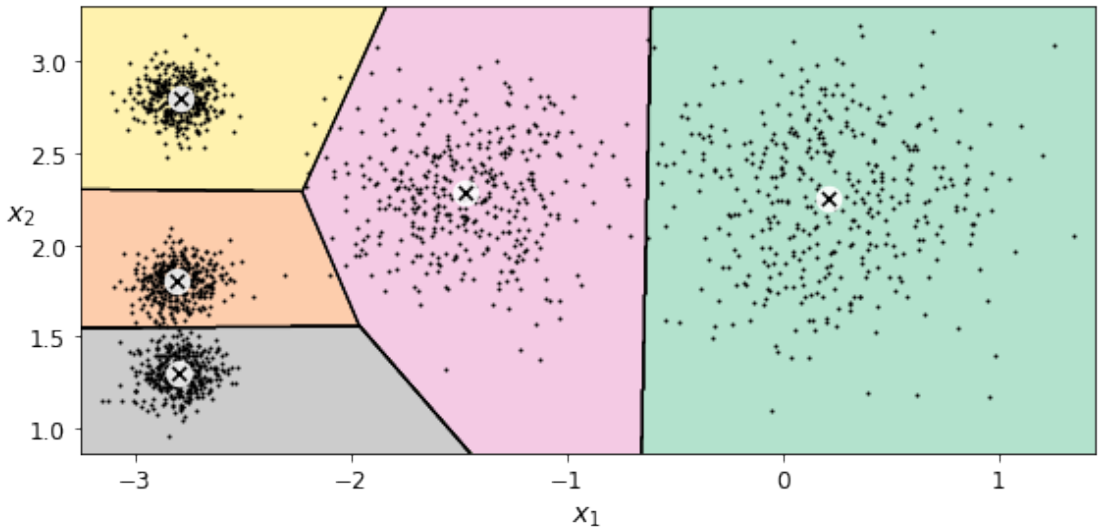
```
array([0, 0, 3, 3], dtype=int32)
```

*Code to reproduce the graphs can be seen at **UUID - #S9C3***

K-Means

Understanding decision boundaries

- The type of K-Means decision boundaries is called a Voronoi polygon
- You can see that the K-Means algorithm **does not behave super well** when the blobs have **different diameters**, since **all it cares about to assign an instance is its distance to the centroid**



*Code to reproduce the graphs can be seen at **UUID - #S9C4***

K-Means

Hard vs. Soft Clustering

- **Hard clustering**: assigning each instance to only a single cluster
- **Soft clustering:** give each instance a score per cluster, which can be:
  - The distance between the instance and the centroid
  - A similarity score such as the Gaussian Radial Basis Function (RBF)
- The `KMeans` class in Sklearn uses the `transform()` method which measures the Euclidean distance from each instance to every centroid
- In the case below, we pass some of the new instances to check how far they are from every centroid:

```
1 for label, distance in zip(kmeans.predict(X_new), kmeans.transform(X_new)):
2     print("Assigned label -> ", label, "| Assigned distance -> ", distance)
```

```
Assigned label ->  0 | Assigned distance ->  [0.32995317 2.81093633 1.49439034 2.9042344  2.88633901]
Assigned label ->  0 | Assigned distance ->  [2.80290755 5.80730058 4.4759332  5.84739223 5.84236351]
Assigned label ->  3 | Assigned distance ->  [3.29399768 1.21475352 1.69136631 0.29040966 1.71086031]
Assigned label ->  3 | Assigned distance ->  [3.21806371 0.72581411 1.54808703 0.36159148 1.21567622]
```

*Code to reproduce the graphs can be seen at **UUID - #S9C5***

K-Means

Hard vs. Soft Clustering

- In this example, the first instance is located at a distance of:
  - 0.3 to centroid 0
  - 2.8 to centroid 1
  - 1.5 to centroid 2
  - 2.9 to centroid 3
  - 2.9 to centroid 4

```
1 for label, distance in zip(kmeans.predict(X_new), kmeans.transform(X_new)):
2     print("Assigned label -> ", label, "| Assigned distance -> ", distance)
```

```
Assigned label ->  0 | Assigned distance ->  [0.32995317 2.81093633 1.49439034 2.9042344  2.88633901]
Assigned label ->  0 | Assigned distance ->  [2.80290755 5.80730058 4.4759332  5.84739223 5.84236351]
Assigned label ->  3 | Assigned distance ->  [3.29399768 1.21475352 1.69136631 0.29040966 1.71086031]
Assigned label ->  3 | Assigned distance ->  [3.21806371 0.72581411 1.54808703 0.36159148 1.21567622]
```

*Code to reproduce the graphs can be seen at **UUID - #S9C5***

K-Means

Hard vs. Soft Clustering

- This technique is very useful for dimensionality reduction
- If we have a high dimensional dataset and we transform it to distance from centroids, we will end up with a **k-dimensional** dataset

```
1 for label, distance in zip(kmeans.predict(X_new), kmeans.transform(X_new)):
2     print("Assigned label -> ", label, "| Assigned distance -> ", distance)
```

```
Assigned label ->  0 | Assigned distance ->  [0.32995317 2.81093633 1.49439034 2.9042344  2.88633901]
Assigned label ->  0 | Assigned distance ->  [2.80290755 5.80730058 4.4759332  5.84739223 5.84236351]
Assigned label ->  3 | Assigned distance ->  [3.29399768 1.21475352 1.69136631 0.29040966 1.71086031]
Assigned label ->  3 | Assigned distance ->  [3.21806371 0.72581411 1.54808703 0.36159148 1.21567622]
```

*Code to reproduce the graphs can be seen at **UUID - #S9C5***

K-Means

Understanding the algorithm

- K-Means starts by placing the centroids randomly
  - It uses k instances at random and uses their location as centroids
- Then K-Means labels the instances, computes distances and updates the centroids
- This is done until centroids stop moving
- The algorithm is guaranteed to converge after a finite number of steps, because the mean squared distance between the instances and their closest centroid can only go down at each step

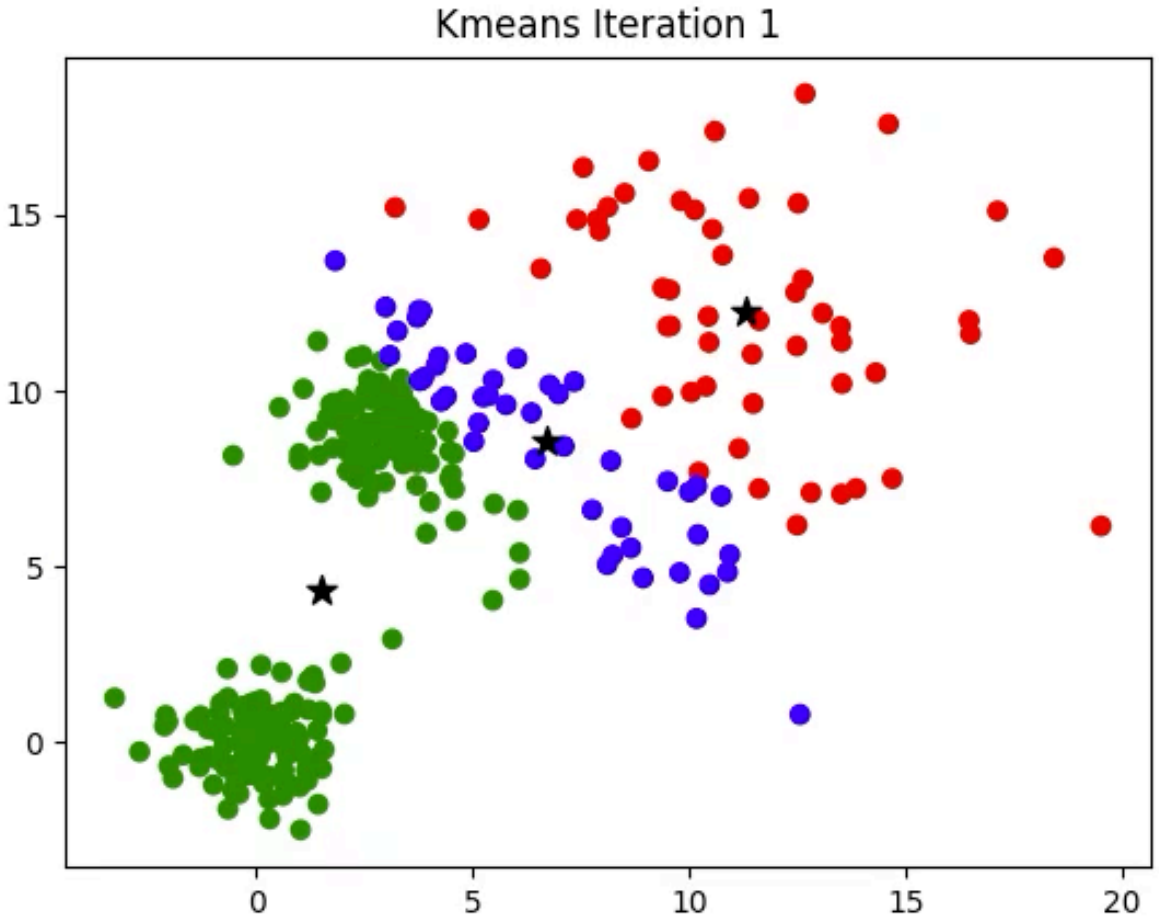
K-Means

Understanding the algorithm

- You can see the algorithm working in next slide
- First centroids are initialized at random
- Then instances are labelled
- Then centroids are updated
- Then instances are relabeled
- Etc.

K-Means

Understanding the algorithm

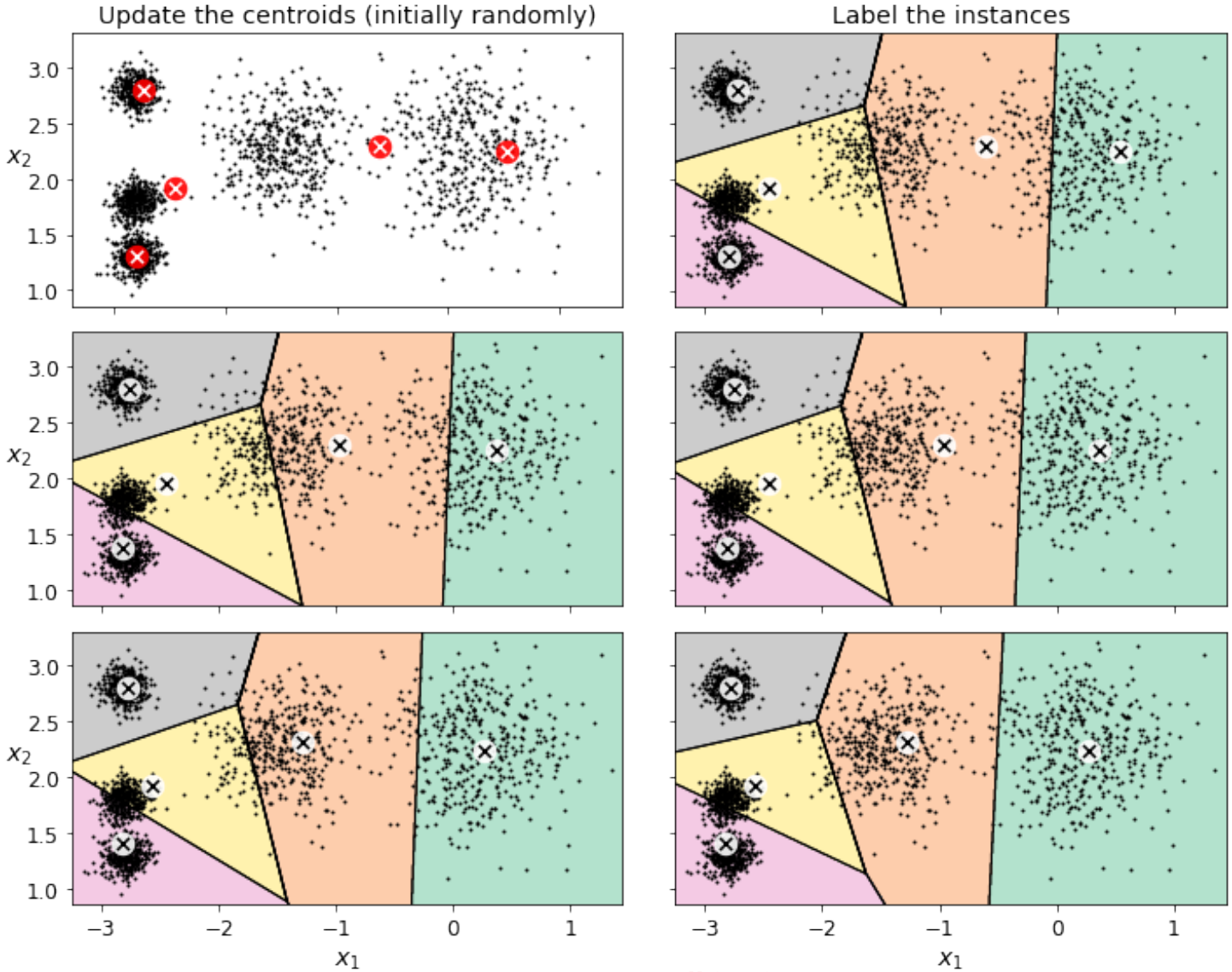


Clustering with K=3

https://towardsdatascience.com/k-means-clustering-from-a-to-z-f6242a314e9a

K-Means

Understanding the algorithm



*Code to reproduce the graphs can be seen at **UUID - #S9C6***

K-Means

Understanding the algorithm

- The computational complexity of K-Means is linear with

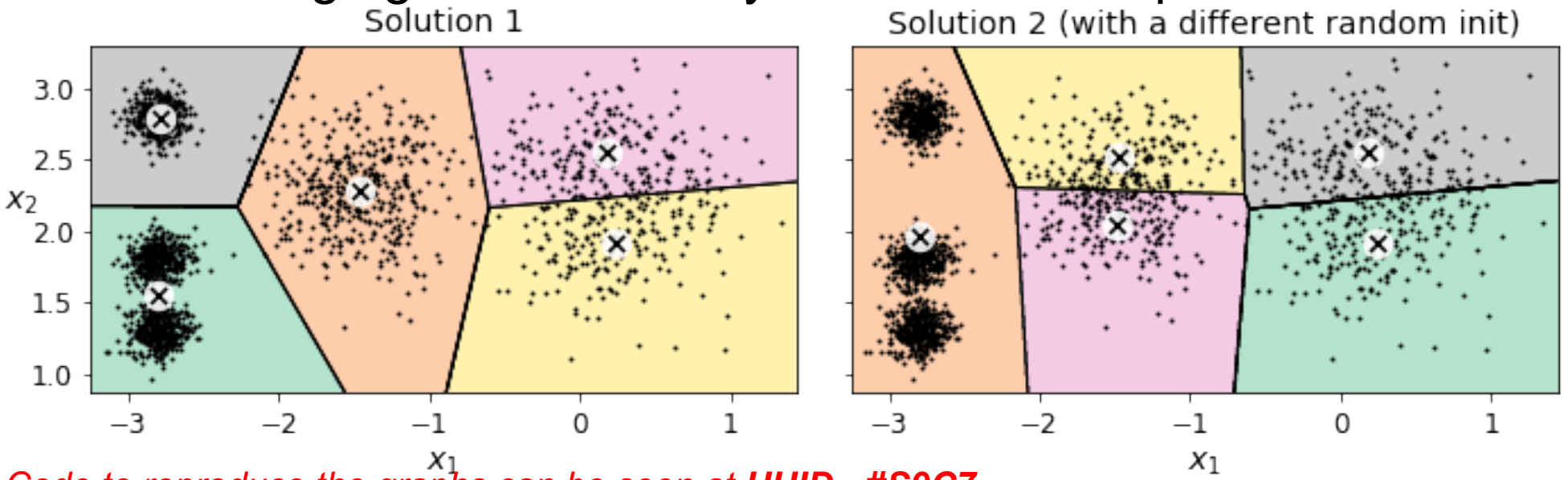
  respects to instances (m),
  number of clusters (k) and the
  number of dimensions (n).

  K-Means is usually one of the fastest clustering algorithms. However, its computational complexity can increase exponentially if data does not have a clustering structure (random circular blobs).

K-Means

The problem of variability

- K-Means is guaranteed to converge, but it might not do it in the right direction
- It can converge to a local optimum
- This all depends on the **centroid initialization**
- The graph below shows two attempts with suboptimal converging due to a faulty initialization step



*Code to reproduce the graphs can be seen at **UUID - #S9C7***

K-Means

Centroid initialization methods

- One good way to initialize the algorithm is to *know approximately* where the centroid should be
- This can be achieved by running another clustering algorithm before
- Then we can set the `init` hyperparameter to the NumPy array containing the list of centroids and set the `n_init` to 1
- Why?

```
good_init = np.array([[-3, 3], [-3, 2], [-3, 1], [-1, 2], [0, 2]])
kmeans = KMeans(n_clusters=5, init=good_init, n_init=1, random_state=42)
kmeans.fit(X)
```

*Code to reproduce the graphs can be seen at **UUID - #S9C8***

K-Means

Centroid initialization methods

- The number of random initializations is controlled by the `n_init` hyperparameter
- Another approach is to use this to run the algorithm multiple times and keep the best solution
- By default, the `n_init` is set to 10, meaning that the whole algorithm runs 10 times when we call `fit()`, and Sklearn keeps the best solution
- The best solution is decided thanks to a performance matrix called the model's *inertia*
  - *Inertia* is the mean squared distance between each instance and its closest centroid

K-Means

Centroid initialization methods

- The `KMeans` class runs the K-Means algorithm `n_init` times and keeps the model with the lowest inertia
  - The model's inertia can be accessed through `the inertia_` instance variable
- The model is controlled through the `score()` which returns the negative inertia:
  - If a predictor is better than another, its `score()` should return a greater value (lowest inertia)

```
1 kmeans.inertia_
```

```
211.5985372581684
```

```
1 kmeans.score(X)
```

```
-211.59853725816856
```

*Code to reproduce the graphs can be seen at **UUID - #S9C8***

K-Means

K-Means++ initialization

- The default initialization in the `KMeans` class is the K-Means++
  - This is a smarter initialization step that tends to select centroids that are distant from one another, and this makes much less likely to converge to a suboptimal solution
  - This algorithm requires extra computation but it is worth because it reduces the number of times the algorithm needs to be run to find the optimal solution

Arthur, Vassilvitskii. (2007)

K-Means

K-Means++: this is how the algorithm works:

- Take one centroid c_1 , chosen uniformly at random from the dataset.
- Compute distances from each observations to c_1. Denote the distance between c_j and the observation m as $d(x_m, c_j)$
- Select the next centroid, c_2 , at random from X with probability

$$\frac{d^2(x_m,c_1)}{\sum_{j=1} d^2(x_j,c_1)}$$

- To choose center j
  - Compute distances from each obervation to each centroid, and assign each observation to its closest centroid
  - For $m = 1, \dots, n$ and $p = 1, \dots, j-1$, select centroid j at random from X with probability

$$\frac{d^2(x_m,c_p)}{\sum_{\{h;x_h \in C_p\}} d^2(x_h,c_p)}$$

    where C_p is the set of all observations closest to centroid c_p and x_m belongs to C_p
  - Hence select each subsequente center with a probability propoertional to the distance from itself to the closest center that you already choose

K-Means

K-Means++: this is how the algorithm works:

- Take one centroid c_1 , chosen uniformly at random from the dataset.
- Compute distances from each observations to c_1. Denote the distance between c_j and the observation m as $d(x_m, c_j)$
- Select the next centroid, c_2 , at random from X with probability

$$\frac{d^2(x_m,c_1)}{\sum_{j=1} d^2(x_j,c_1)}$$

Repeat this step until k centroids are chosen

- To choose center j
  - Compute distances from each observation to each centroid, and assign each observation to its closest centroid
  - For $m = 1, \ldots, n$ and $p = 1, \ldots, j-1$, select centroid j at random from X with probability

$$\frac{d^2(x_m,c_p)}{\sum_{\{h;x_h \in C_p\}} d^2(x_h,c_p)}$$

    where C_p is the set of all observations closest to centroid c_p and x_m belongs to C_p
  - Hence select each subsequente center with a probability proportional to the distance from itself to the closest center that you already choose

K-Means

K-Means++ initialization

- By default Sklearn uses `k-means++` and you will rarely will need to set `init` to random

```
1 KMeans()
```

```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
    n_clusters=8, n_init=10, n_jobs=None, precompute_distances='auto',
    random_state=None, tol=0.0001, verbose=0)
```

*Code to reproduce the graphs can be seen at **UUID - #S9C9***

K-Means

Accelerated K-Means

- The Accelerated K-Means can be significantly accelerated by avoiding many unnecessary distance calculations
- This achieved by exploiting the triangle inequality (given three points A, B and C, the distance AC is always such that AC ≤ AB + BC) and by keeping track of lower and upper bounds for distances between instances and centroids
- This algorithm is used by default by the `KMeans` class. If you want to change to the original algorithm (you will almost never need to do that), you can use `full` within the `algorithm` hyperparameter

Elkan, (2003)

K-Means

Accelerated K-Means

```
1 %timeit -n 50 KMeans(algorithm="elkan").fit(X)
```

```
50 loops, best of 3: 86.4 ms per loop
```

```
1 %timeit -n 50 KMeans(algorithm="full").fit(X)
```

```
50 loops, best of 3: 125 ms per loop
```

*Code to reproduce the graphs can be seen at **UUID - #S9C10***

K-Means

Mini-batch K-Means

- This algorithm is capable of using mini-batches, moving the centroids slightly at each iteration
- This eventually speeds up the algorithm by a factor of three or four and makes it possible to cluster huge datasets that do not fit into memory
- It is implemented as `MiniBatchKMeans` class in Sklearn

```
1 from sklearn.cluster import MiniBatchKMeans
```

```
1 minibatch_kmeans = MiniBatchKMeans(n_clusters=5, random_state=42)
2 minibatch_kmeans.fit(X)
```

```
MiniBatchKMeans(batch_size=100, compute_labels=True, init='k-means++',
                init_size=None, max_iter=100, max_no_improvement=10,
                n_clusters=5, n_init=3, random_state=42,
                reassignment_ratio=0.01, tol=0.0, verbose=0)
```

```
1 minibatch_kmeans.inertia_
```

```
211.93186531476775
```

*Code to reproduce the graphs can be seen at **UUID - #S9C11***

Sculley, (2010)

K-Means

Mini-batch K-Means

Using Mini-Batch with large datasets:

Go to Notebook ***UUID - #S9C11***

K-Means

K-Means vs Mini-Batch K-Means

- Even if Mini-Batch K-Means is way faster than bare K-Means, its inertia is usually worse

```
1 %timeit KMeans(n_clusters=5).fit(X)
```

```
10 loops, best of 3: 46.5 ms per loop
```

```
1 %timeit MiniBatchKMeans(n_clusters=5).fit(X)
```

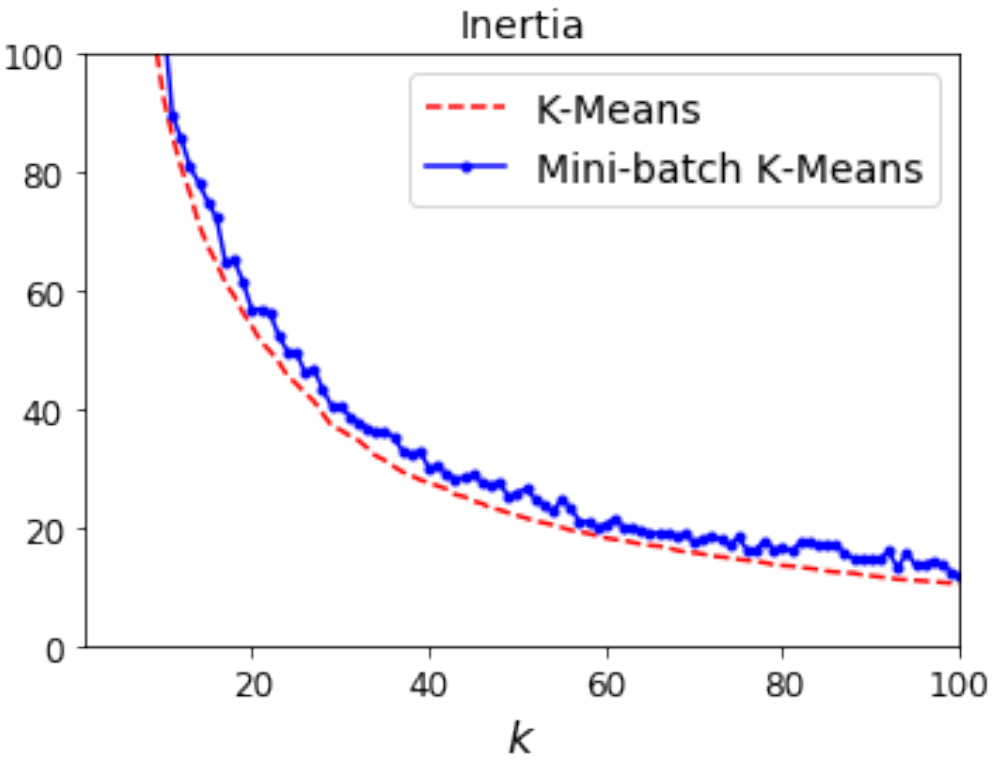
```
10 loops, best of 3: 28.5 ms per loop
```

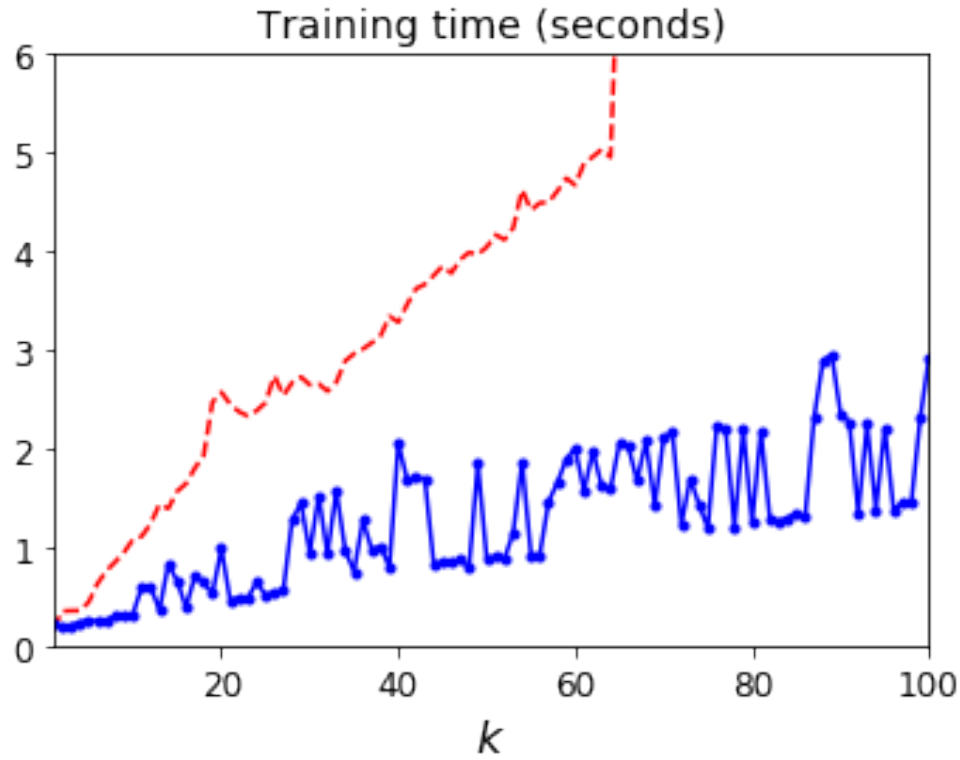
*Code to reproduce the graphs can be seen at **UUID - #S9C12***

K-Means

K-Means vs Mini-Batch K-Means

- Even if Mini-Batch K-Means is way faster than bare K-Means, its inertia is usually worse





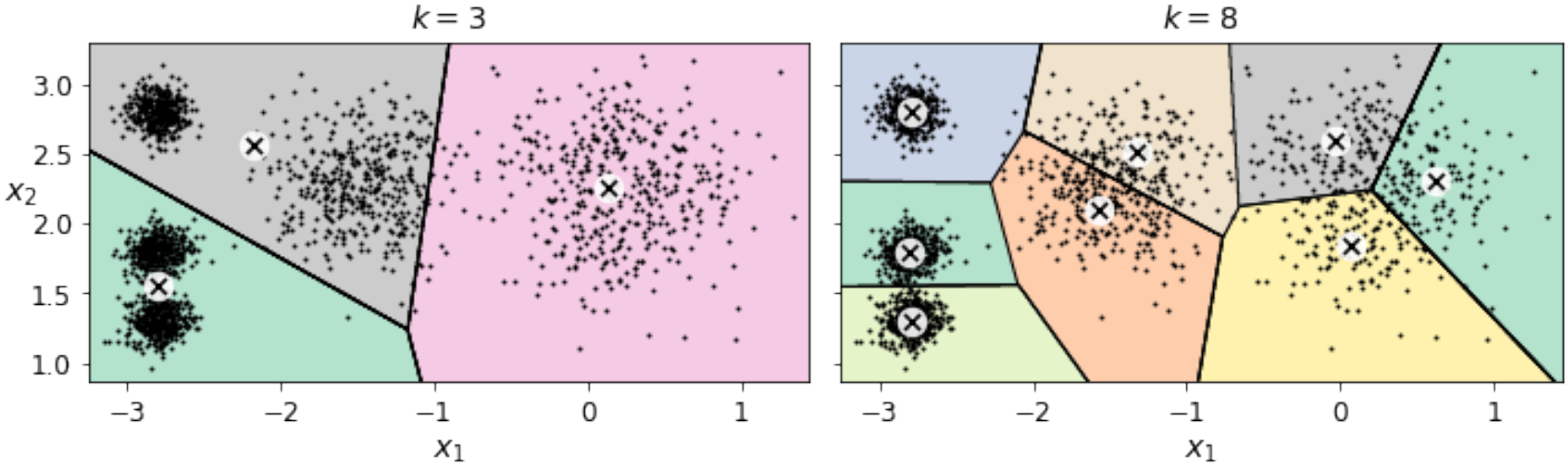
*Code to reproduce the graphs can be seen at **UUID - #S9C12***

K-Means

Finding optimal number of clusters

- How do we pick the number of clusters, for this blob case it was kind of clear that we had to choose 5 clusters

- Should we just pick the model with the lowest inertia?



*Code to reproduce the graphs can be seen at **UUID - #S9C13***

K-Means

Finding optimal number of clusters

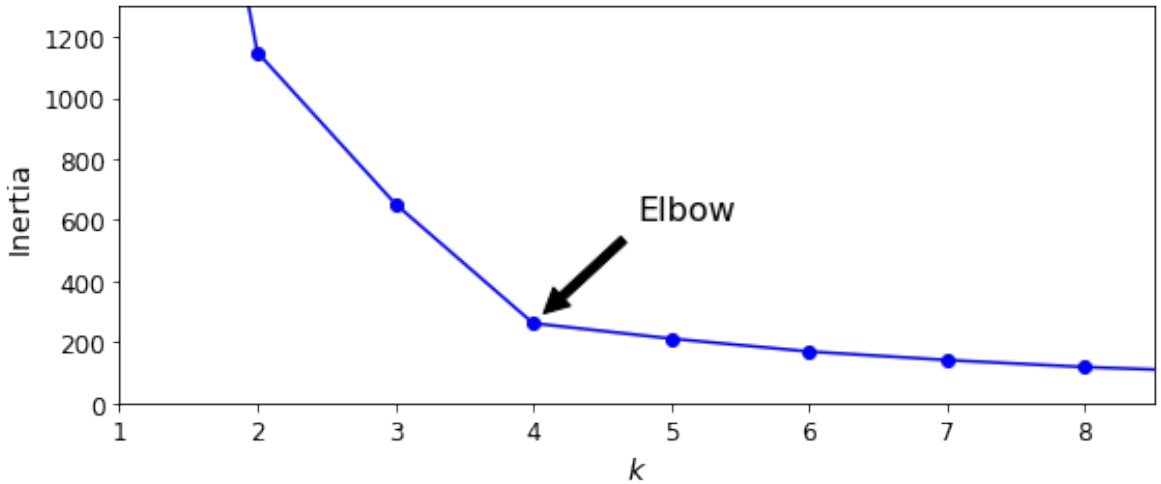
- Unfortunately, inertia is not a good performance metric when trying to choose the number of clusters
- This is because inertia tends to get lower as we increase the number of clusters
  - The more clusters there are, the closer each instance will be to its closest centroid, and thus the lower the inertia will be

```
1 kmeans_k3.inertia_
```

```
653.2167190021553
```

```
1 kmeans_k8.inertia_
```

```
118.41983763508077
```

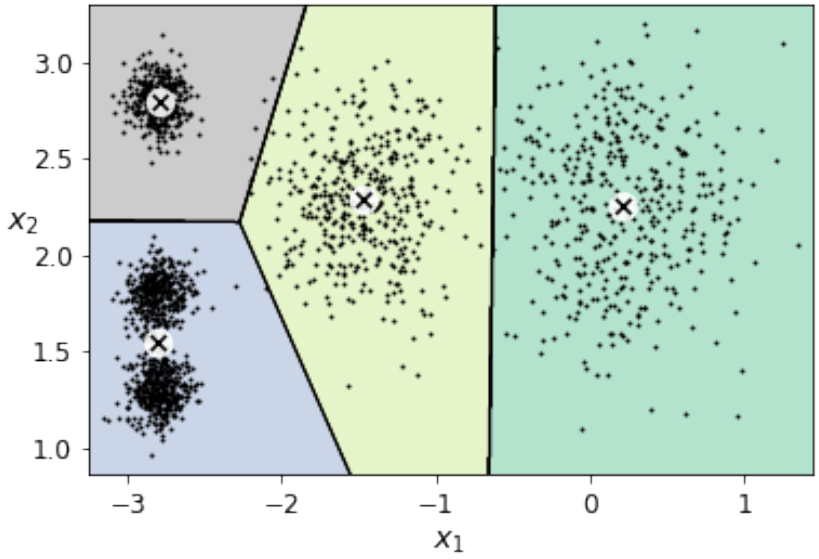


*Code to reproduce the graphs can be seen at **UUID - #S9C13***

K-Means

Finding optimal number of clusters

- According to the commonly known technique of finding the optimal “elbow”, if we didn’t know our dataset, we will pick 4 clusters in this case
- But we know that k=4 is not ideal, so we will need to choose another methods since the elbow is quite coarse



*Code to reproduce the graphs can be seen at **UUID - #S9C13***

K-Means

Finding optimal number of clusters – Silhouette Score

- The Silhouette approach is a rather more computationally expensive method but is more precise
- It computes the mean ***silhouette coefficient*** over all the instances
- An instance's silhouette coefficient is equal to $(b-a)/\max(a,b)$
  - where a is the mean distance to the other instances in the same cluster (it is the mean intra-cluster distance)
  - b is the mean nearest-cluster distance, that is the mean distance to the instances of the next closest cluster (defined as the one that minimizes b , excluding the instance's own cluster).
- The silhouette coefficient can vary between -1 and +1:
  - A **coefficient close to +1** means that the instance is well inside its own cluster and far from other clusters,
  - A **coefficient close to 0** means that it is close to a cluster boundary
  - A **coefficient close to -1** means that the instance may have been assigned to the wrong cluster

*Code to reproduce the graphs can be seen at **UUID - #S9C13***

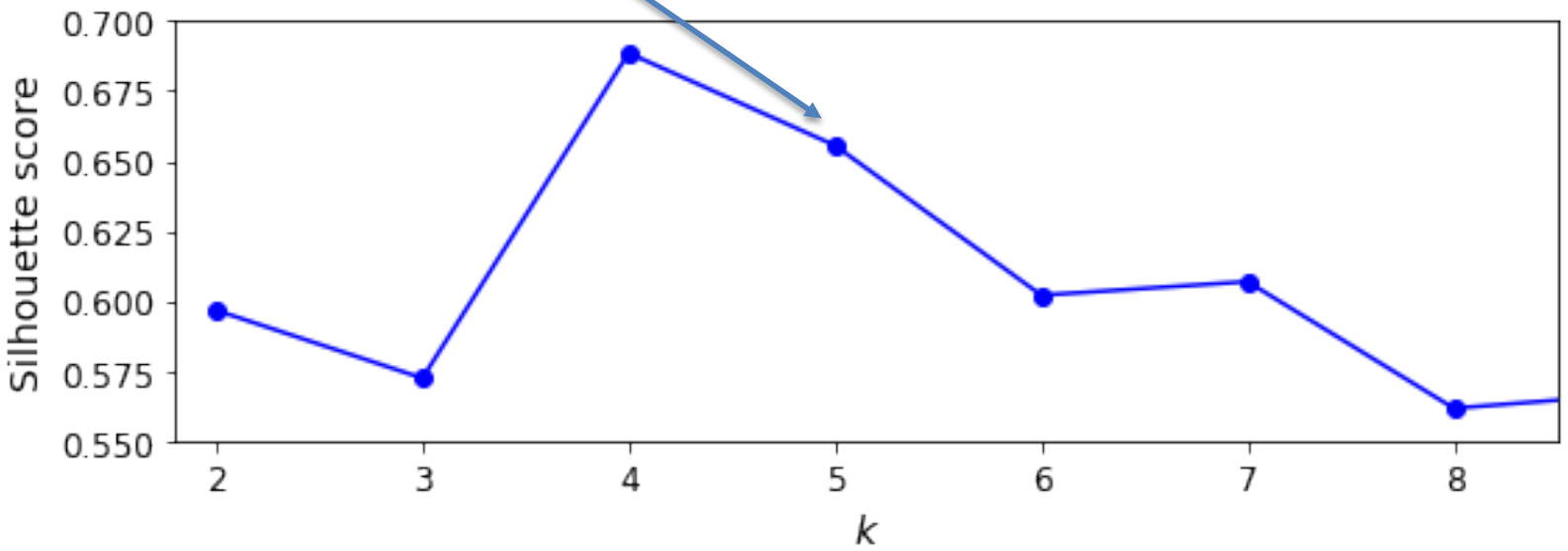
K-Means

Finding optimal number of clusters – Silhouette Score

```
1 from sklearn.metrics import silhouette_score
```

```
1 silhouette_score(X, kmeans.labels_)
```

```
0.655517642572828
```

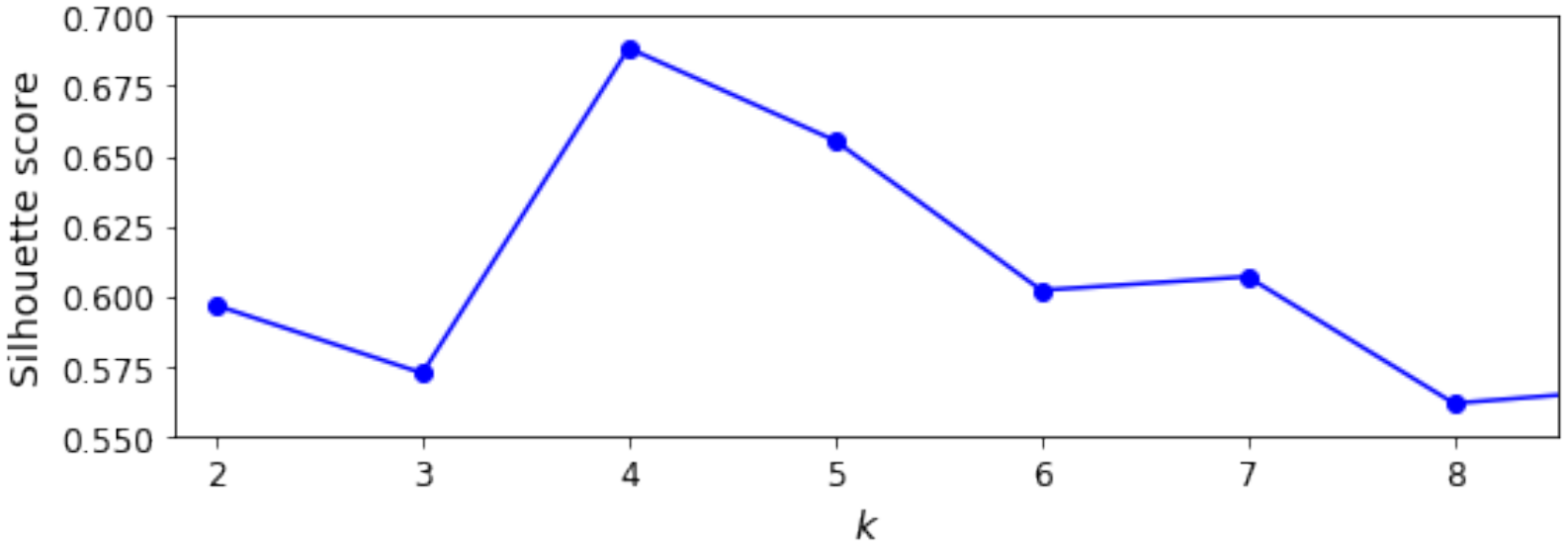


*Code to reproduce the graphs can be seen at **UUID - #S9C13***

K-Means

Finding optimal number of clusters – Silhouette Score

- The Silhouette Score is more informative than the mere inertia plot
- This shows that perhaps k=4 is the best but k=5 is very good as well and considerably above 6 and 7 which could not be noticed before



*Code to reproduce the graphs can be seen at **UUID - #S9C13***

K-Means

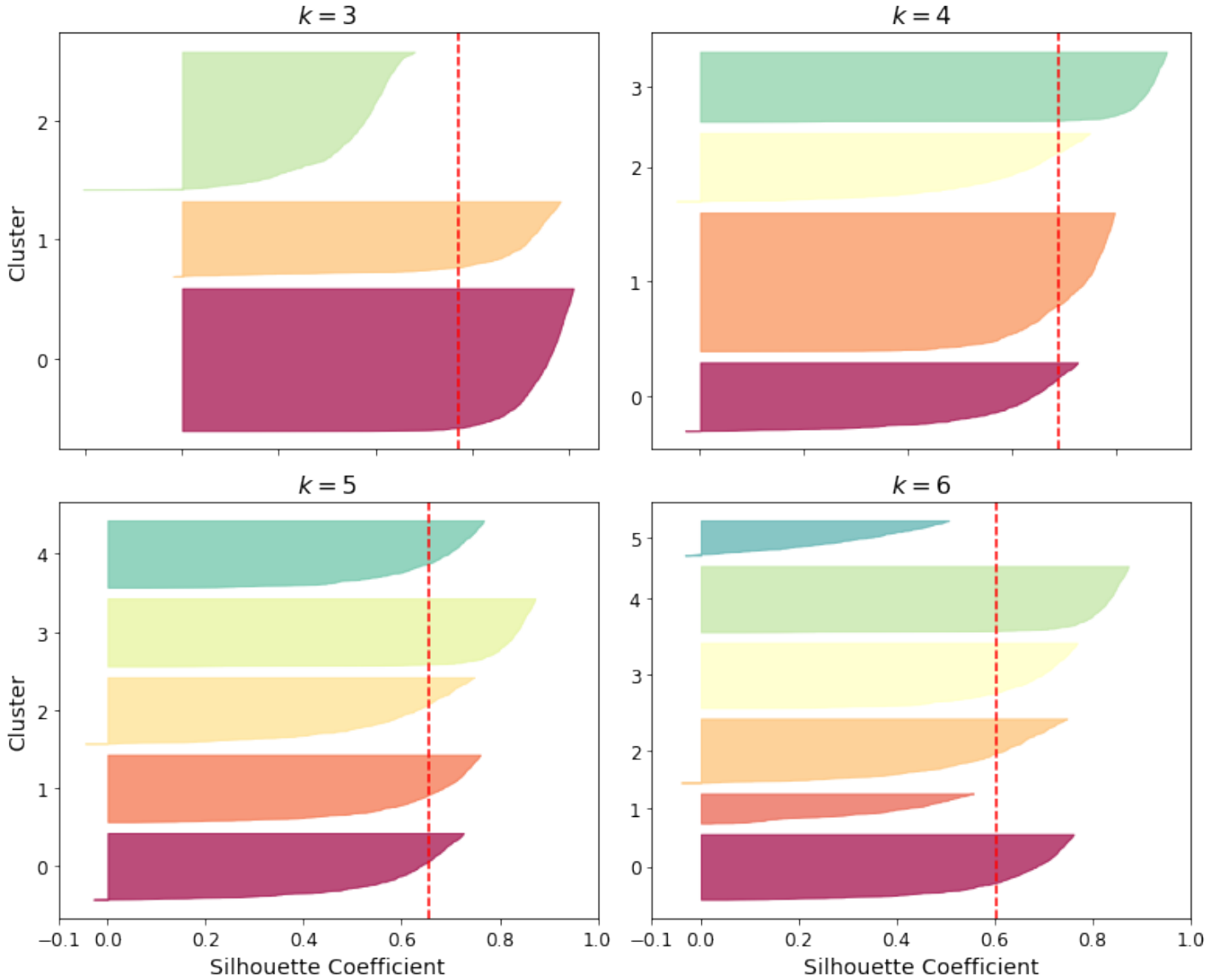
Finding optimal number of clusters – Silhouette Diagram

- There is yet a more informative visualization called the *Silhouette Diagram*
- This is achieved when you plot every instance's silhouette coefficient, sorted by the cluster they are assigned to and by the value of the coefficient
- The **shape's height** defines the number of instances the cluster contains
- The **width** represents the sorted silhouette coefficients of the instances in the cluster
- The **dashed line** indicates mean silhouette coefficient
  - When most of the instances are to the left of this dashed line (closer to 0), it means that the cluster is rather bad, since its instances are much too close to other clusters

*Code to reproduce the graphs can be seen at **UUID - #S9C13***

K-Means

Finding optimal number of clusters – Silhouette Score



*Code to reproduce the graphs can be seen at **UUID - #S9C13***

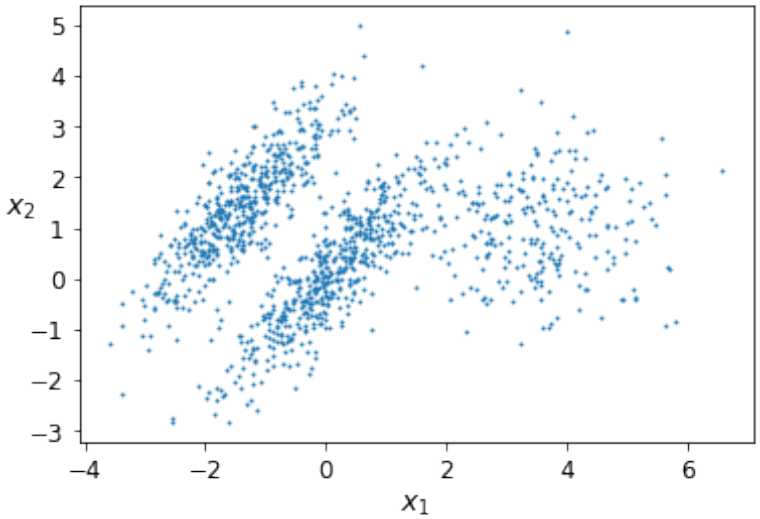
Unsupervised Learning

Limits of K-Means

Limits of K-Means

Some cons

- K-Means is an awesome algorithm, but is not perfect
  - We need to run the algorithm several times to avoid suboptimal solutions
  - We need to specify the number of clusters and all the "exploratory cluster" decisions seem a bit handcrafted
  - K-Means does not behave well when clusters have different sizes, densities or non-spherical shapes

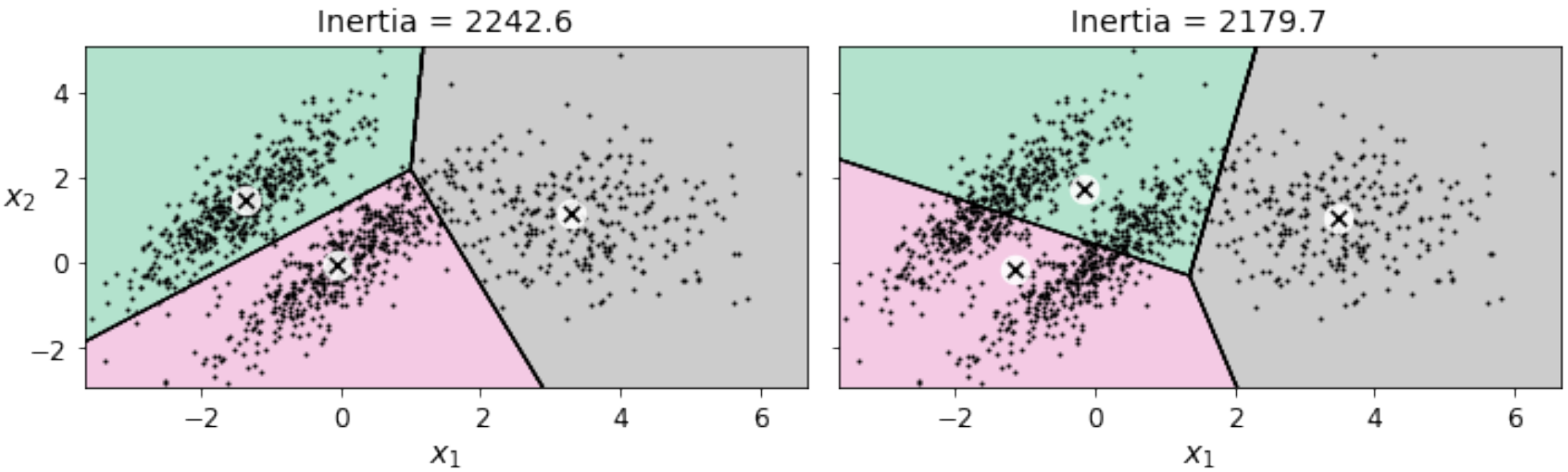


*Code to reproduce the graphs can be seen at **UUID - #S9C14***

Limits of K-Means

Some cons

- Neither of the solutions proposed is any good, the solution to the right is horrible… even if inertia is lower!!
- On these type of datasets… GMMs will perform better as we will see
- However… there is one trick…



*Code to reproduce the graphs can be seen at **UUID - #S9C14***

Limits of K-Means

Some cons

- Whenever you have such datasets, try to scale it before passing the `KMeans`, using `StandardScaler()` or such. This does not ensure that clusters will be spherical and equally shaped, but it will certainly improve things.

BTS | Barcelona Technology School

MBDS 2020-2021

Usages of K-Means

Image Segmentation

K-Means: Image Segmentation

Color Segmentation

- The real state of the art in image segmentation now it is done with CNNs
- However K-Means are very good for a first exploratory color segmentation
  - Simply assigning pixels to the same segment if they have a similar color
  - This sometimes is sufficient if we want to analyze satellite images to measure how much total forest area there is in a region
- We load a sample image and use Matplotlib to read it:

```
1 from matplotlib.image import imread
2 image = imread(os.path.join(images_path, filename))
3 image.shape
```

```
(533, 800, 3)
```

*Code to reproduce the graphs can be seen at **UUID - #S9C15***

K-Means: Image Segmentation

Color Segmentation

- Images always come in a 3D array, we need to reshape it or flatten it in order to apply Kmeans
- In this case we flatten each channel getting a 1D array per channel

```
1 X = image.reshape(-1, 3)
2 print(X.shape)
3 kmeans = KMeans(n_clusters=8, random_state=42).fit(X)
4 segmented_img = kmeans.cluster_centers_[kmeans.labels_]
5 segmented_img = segmented_img.reshape(image.shape)
```

```
(426400, 3)
```

*Code to reproduce the graphs can be seen at **UUID - #S9C15***

K-Means: Image Segmentation

Color Segmentation

- Notice how after 4 clusters the ladybug's flashy red gets merged with the colors of environment. This is because K-Means prefers clusters of similar sizes, and the ladybug is very small so K-Means fails to allocate a cluster to it

*Code to reproduce the graphs can be seen at **UUID - #S9C15***

Usages of K-Means

Preprocessing

K-Means: Preprocessing

Massaging the Data

- Preprocessing stands for the process of “massaging the data” before running a classifier
- Preprocessing can be understood as a dimensionality reduction technique which can be tackled through clustering, i.e. going from *n-dimensions* in the dataset to *k-dimensions*
- Let’s do an example building a pipeline in Sklearn using KMeans for clustering and then running a classifier to it

Go to notebook – **UUID #S9E1**

Usages of K-Means

Semi-supervised learning

K-Means: Semi-supervised Learning

Going towards real world datasets

- Semi-supervised learning is useful when we have plenty of unlabeled instances and very few labeled instances, like it happened at the example at the beginning
- In this case we can use techniques of finding the **representative images** and doing **label propagation** to acquire similar accuracy results than those on larger datasets, but using much less labelled data

Go to notebook – **UUID #S9C16**

BTS | Barcelona Technology School

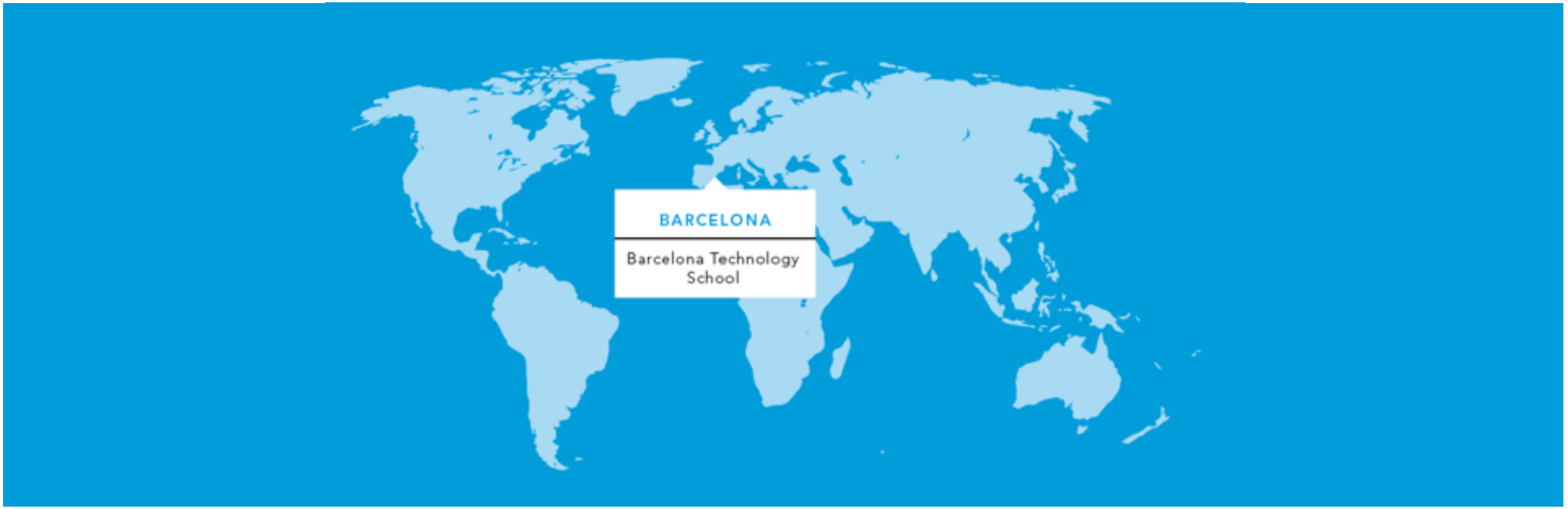
MBDS 2020-2021

RECAP

Resources

Important resources

- Lex Friedman Series (MIT)
- Sklearn docs
- Papers linked through the algorithm explanaition
- Jake VanderPlas, “Python Data Science Handbook”
- Aurelien Geron, “Hands-on machine learning with scikit-learn, Keras & Tensorflow”


BARCELONA
Barcelona Technology School