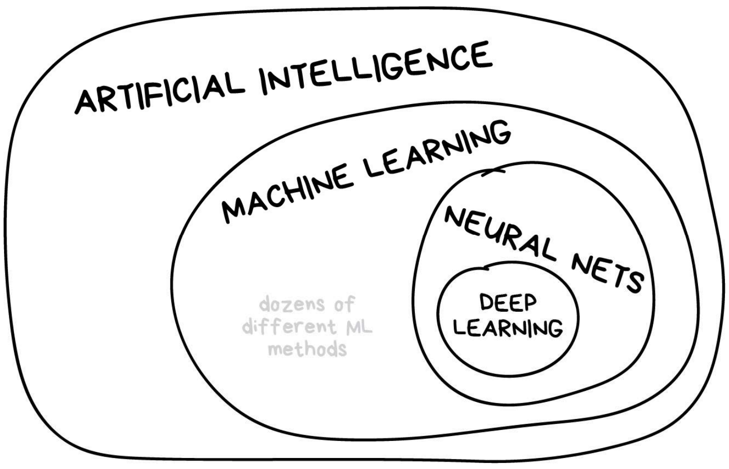
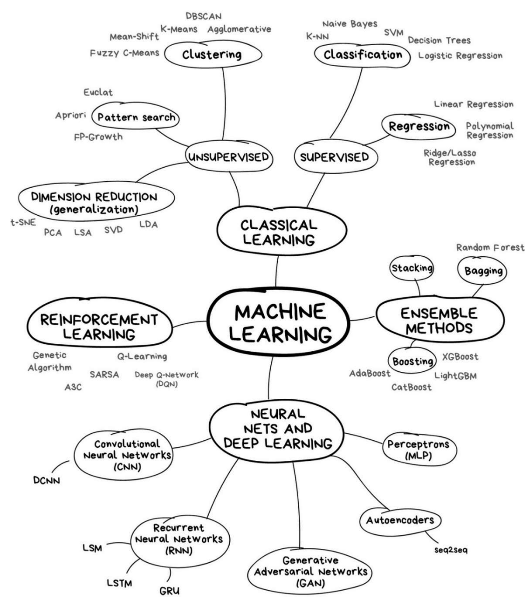
UNSUPERVISED LEARNING TECHNIQUES

MODULE 8

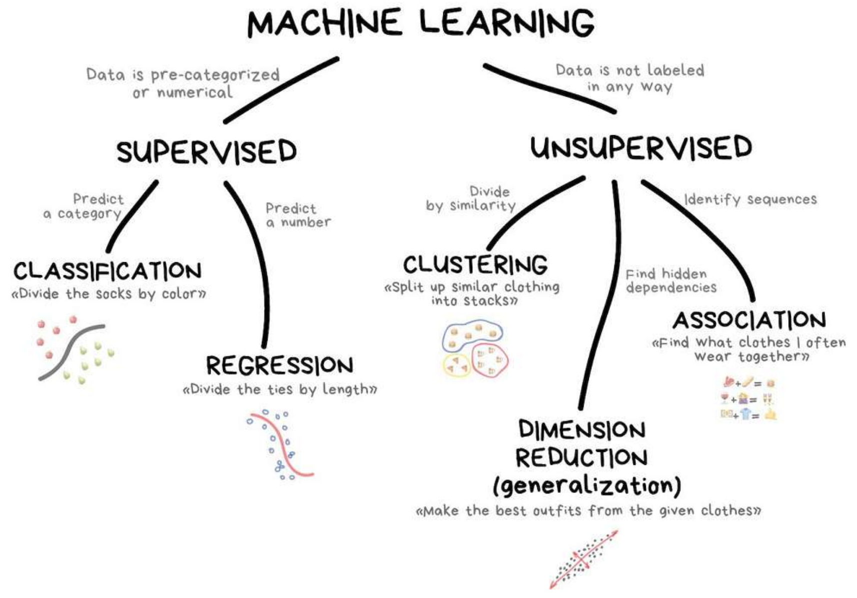
Basic Keywords



Basic Keywords for ML



Basic Keywords for ML


MACHINE LEARNING
Data is pre-categorized or numerical
Data is not labeled in any way
SUPERVISED
UNSUPERVISED
Predict a category
Predict a number
CLASSIFICATION
«Divide the socks by color»
REGRESSION
«Divide the ties by length»
Divide by similarity
Identify sequences
Find hidden dependencies
CLUSTERING
«Split up similar clothing into stacks»
ASSOCIATION
«Find what clothes I often wear together»
DIMENSION REDUCTION (generalization)
«Make the best outfits from the given clothes»

Three components of machine learning

> The goal of machine learning is to **predict results based on incoming data**.

– **DATA:** Want to detect spam? Get samples of spam messages. Want to forecast stocks? Find the price history. Want to find out user preferences? Parse their activities on Facebook (Stop Zuckerberg! pls stop collectiong and do not sell our data to advertisers)

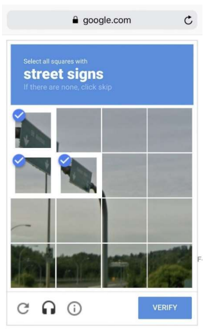
- There are two main ways to get the data
  – Manual (take more time but more accurate)
  – Automatic (cheaper)

Three components of machine learning

> The goal of machine learning is to **predict results based on incoming data**.

– **DATA:** collect a good collection of data (usually called a dataset)

– They are so important that companies may even reveal their algorithms, <u>but rarely datasets</u>.



Three components of machine learning

> The goal of machine learning is to **predict results based on incoming data**.

– **FEATURES:** Also known as parameters or variables. Those could be car mileage, user's gender, stock price, word frequency in the text. In other words, these are the factors for a machine to look at.

- We use sophisticated «Feaure Extractors» to find right features of data

Three components of machine learning

> The goal of machine learning is to **predict results based on incoming data**.

– **ALGORITHMS:** Any problem can be solved differently. The method you choose affects the precision, performance, and size of the final model.

- If the data is crappy, even the best algorithm won't help.
- Sometimes it's referred as "garbage in – garbage out".
- So don't pay too much attention to the percentage of accuracy, try to acquire more **data** first.

Introduction

- > Although most of the applications of Machine Learning today are based on supervised learning
  - – That is why most of the investments go to there
- > The vast majority of the available data is actually unlabeled: we have the input features **X**, but we do not have the labels **y**.

Introduction

- > Yann LeCun famously said that

  “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”
- > In other words, there is a huge potential in unsupervised learning that we have only barely started to sink our teeth into.

Introduction

> Example:

- Say you want to create a system that will take a few pictures of each item on a manufacturing production line and detect which items are defective
- You can fairly easily create a system that will take pictures automatically, and this might give you thousands of pictures every day

Introduction

> Example:

- Say you want to create a system that will take a few pictures of each item on a manufacturing production line and detect which items are defective
- You can fairly easily create a system that will take pictures automatically, and this might give you thousands of pictures every day

- You can then build a reasonably large dataset in just a few weeks.

Introduction

> Example:
  - Say you want to create a system that will take a few pictures of each item on a manufacturing production line and detect which items are defective
  - You can fairly easily create a system that will take pictures automatically, and this might give you thousands of pictures every day
  - You can then build a reasonably large dataset in just a few weeks.
  - But wait, there are no labels!

Introduction

> If you want to train a regular binary classifier that will predict whether an item is defective or not, you will need to label every single picture as "defective" or "normal".

> This will generally require human experts to sit down and manually go through all the pictures.

> This is a long, costly and tedious task, so it will usually only be done on a small subset of the available pictures.

> As a result, the labeled dataset will be quite small, and the classifier's performance will be disappointing

Introduction

> If you want to train a regular binary classifier that will predict whether an item is defective or not, you will need to label every single picture as “defective” or “normal”.

> This will generally require human experts to sit down and manually go through all the pictures.

> This is a long, costly and tedious task, so it will usually only be done on a small subset of the available pictures.

> As a result, the labeled dataset will be quite small, and the classifier’s performance will be disappointing

> Moreover, every time the company makes any change to its products, the whole process will need to be started over from scratch

Content

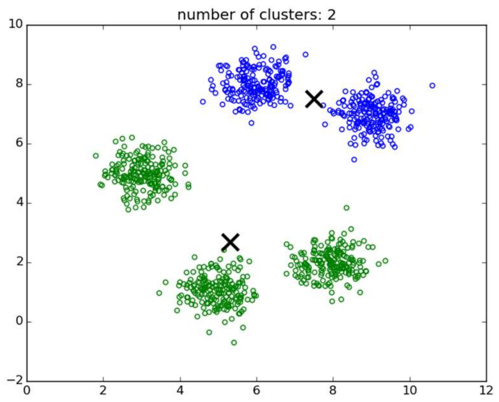
> In Module 8, we looked at the most common unsupervised learning task: dimensionality reduction.

> In this module, we will look at a few more unsupervised learning tasks and algorithms:

- *Clustering*
- *Anomaly detection*
- *Density estimation*

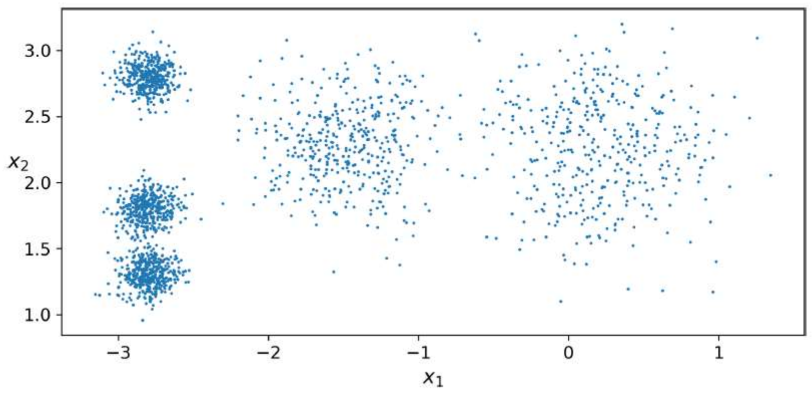
K-Means

> *An unlabeled dataset composed of five blobs of instances*



K-Means

> *An unlabeled dataset composed of five blobs of instances*



K-Means

- Let’s train a K-Means clusterer on this dataset.
- It will try to find each blob’s center and assign each instance to the closest blob:

```
from sklearn.cluster import KMeans
k = 5
kmeans = KMeans(n_clusters=k)
y_pred = kmeans.fit_predict(X)
```

- Note that you have to specify the number of clusters *k* that the algorithm must find.
- In this example, it is pretty obvious from looking at the data that *k* should be set to 5
- In general it is not that easy.

K-Means

- Each instance was assigned to one of the 5 clusters.
- In the context of clustering, an instance’s *label* is the index of the cluster that this instance gets assigned to by the algorithm
- This is not to be confused with the class labels in classification

```
>>> y_pred
array([4, 0, 1, ..., 2, 1, 0], dtype=int32)
>>> y_pred is kmeans.labels_
True
```

K-Means

> We can also take a look at the 5 centroids that the algorithm found:

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

> You can easily assign new instances to the cluster whose centroid is closest:

```
>>> X_new = np.array([[0, 2], [3, 2], [-3, 3], [-3, 2.5]])
>>> kmeans.predict(X_new)
array([1, 1, 2, 2], dtype=int32)
```

Hard Clustering and Soft Clustering

> Assigning each instance to a single cluster is called *hard clustering*

> Assigning each instance to a score per cluster is called *soft clustering*

– the score can be
  - the distance between the instance and the centroid, or
  - A similarity score (or affinity) such as the Gaussian Radial Basis Function

K-Means

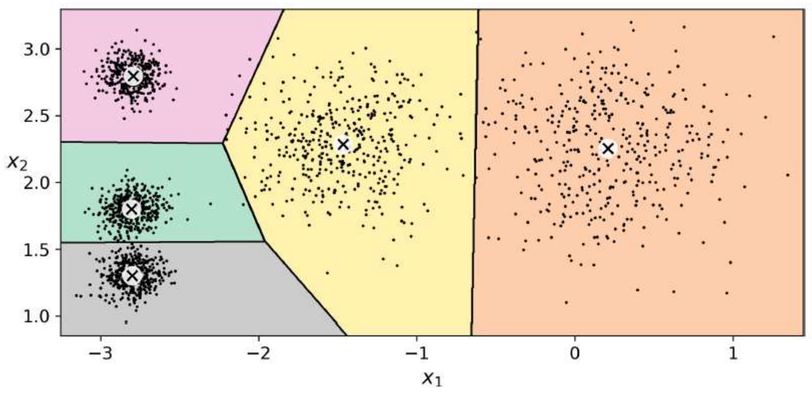
> In the KMeans class, the **`transform()`** method measures the distance from each instance to every centroid

```
>>> kmeans.transform(X_new)
array([[2.81093633, 0.32995317, 2.9042344 , 1.49439034, 2.88633901],
       [5.80730058, 2.80290755, 5.84739223, 4.4759332 , 5.84236351],
       [1.21475352, 3.29399768, 0.29040966, 1.69136631, 1.71086031],
       [0.72581411, 3.21806371, 0.36159148, 1.54808703, 1.21567622]])
```

> If you have a high-dimensional dataset and you transform it this way, you end up with a *k*-dimensional dataset

> This can be a very efficient non-linear dimensionality reduction technique

K-Means decision boundaries



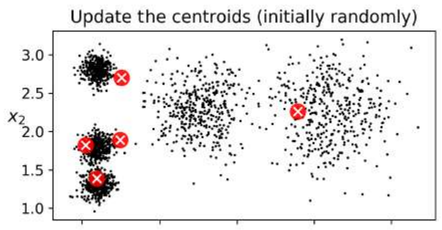
The K-Means Algorithm

> How does the algorithm work?

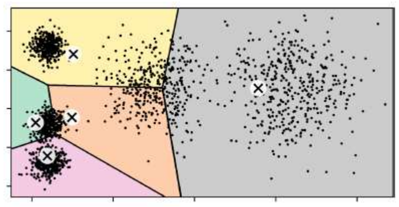
The K-Means Algorithm

> How does the algorithm work?
> Simple: update the centroids, label the instances, update the centroids, and so on…

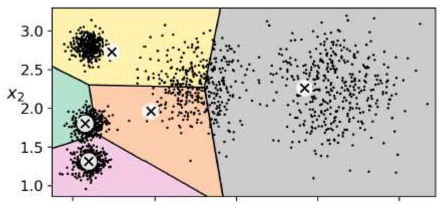
The K-Means Algorithm



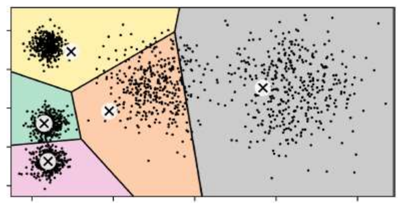
The K-Means Algorithm



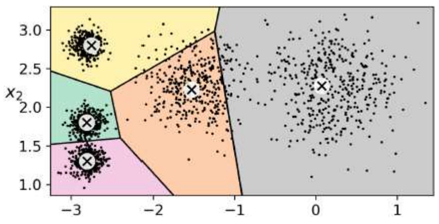
The K-Means Algorithm



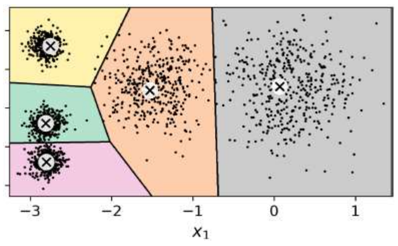
The K-Means Algorithm



The K-Means Algorithm



The K-Means Algorithm



The K-Means Algorithm

> The computational complexity of the algorithm is generally linear with regards to the number of instances ***m***, the number of clusters ***k*** and the number of dimensions ***n***

> However, this is only true when the data has a clustering structure.

> If it does not, then in the worst case scenario the complexity can increase exponentially with the number of instances.

> In practice, however, this rarely happens, and *K-Means is generally one of the fastest clustering algorithms*.

The K-Means Algorithm

> Although the algorithm is guaranteed to converge

> It may not converge to the right solution

– Converge to a local optimum

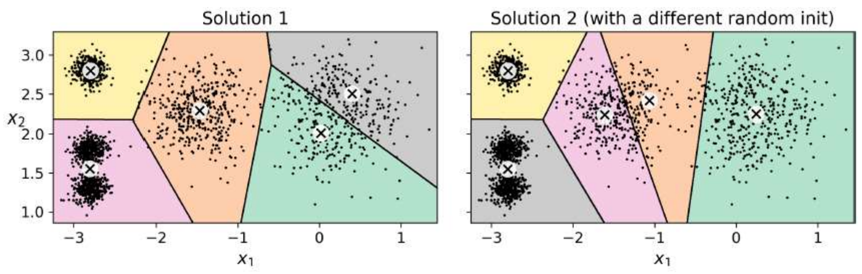
– This depends on the centroid initialization

The K-Means Algorithm

> Although the algorithm is guaranteed to converge

> It may not converge to the right solution

- Converge to a local optimum
- This depends on the centroid initialization



Centroid Initialization Methods

1. If you happen to know approximately where the centroids should be, then you can set the `init` hyperparameter to a **NumPy** array containing the list of centroids, and set `n_init` to 1:

```
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)
```

Centroid Initialization Methods

2. Another solution is to run the algorithm multiple times with different random initializations and keep the best solution.
   - This is controlled by the n_init hyperparameter:
   - Default value is equal to 10
   - The whole algorithm described earlier actually runs 10 times when you call fit(), and Scikit-Learn keeps the best solution.
   - How exactly does it know which solution is the best?

Centroid Initialization Methods

2. Another solution is to run the algorithm multiple times with different random initializations and keep the best solution.
   - This is controlled by the n_init hyperparameter:
   - Default value is equal to 10
   - The whole algorithm described earlier actually runs 10 times when you call fit(), and Scikit-Learn keeps the best solution.
   - How exactly does it know which solution is the best?

     - It uses a performance metric: the model’s inertia
     - The mean squared distance between each instance and its closest centroid.

Model’s Inertia

> A model’s inertia is accessible via the `inertia_` instance variable:

```
>>> kmeans.inertia_
211.59853725816856
```

> The score() method returns the negative inertia.

> Why negative?
  – A predictor’s score() method must always respect the "*great is better*" rule

```
>>> kmeans.score(X)
-211.59853725816856
```

K-Means++

> An important improvement to the K-Means algorithm
  – Smarter initialization step that tends to select centroids that are distant from one another
  – Makes the K-Means algorithm much less likely to converge to a suboptimal solution.
  – The additional computation required for the smarter initialization step is well worth it since it makes it possible to drastically reduce the number of times the algorithm needs to be run to find the optimal solution.

K-Means++

> K-Means++ initialization algorithm:
- Take one centroid $\mathbf{c}^{(1)}$, chosen uniformly at random from the dataset.
- Take a new centroid $\mathbf{c}^{(i)}$, choosing an instance $\mathbf{x}^{(i)}$ with probability:

$$D\left(\mathbf{x}^{(i)}\right)^2 \quad / \quad \sum_{j=1}^{m} D\left(\mathbf{x}^{(j)}\right)^2$$

- $D(\mathbf{x}^{(i)})$ is the distance between the instance $\mathbf{x}^{(i)}$ and the closest centroid that was already chosen.
- This probability distribution ensures that instances further away from already chosen centroids are much more likely be selected as centroids.
- Repeat the previous step until all *k* centroids have been chosen.
- `KMeans` class uses this initialization method by default

Accelerated K-Means

> Another important improvement to the K-Means algorithm

> It considerably accelerates the algorithm by avoiding many unnecessary distance calculations achieved by
- exploiting the triangle inequality
- keeping track of lower and upper bounds for distances between instances and centroids

> This is the algorithm used by default by the `KMeans` class

Mini-batch K-Means

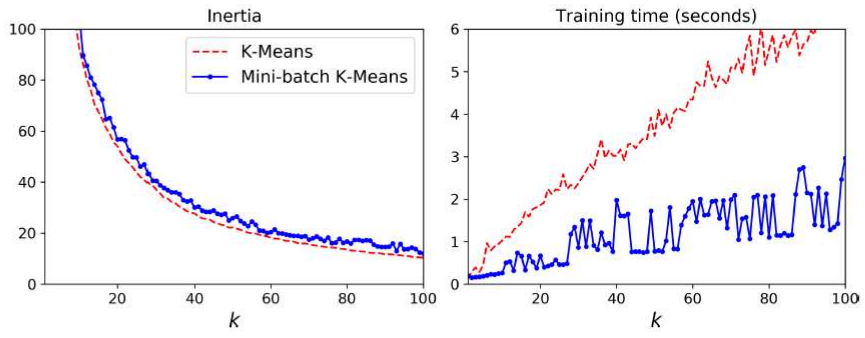
> Instead of using the full dataset at each iteration, the algorithm is capable of using mini-batches
> Moves the centroids just slightly at each iteration.
> This speeds up the algorithm typically by a factor of 3 or 4 and makes it possible to cluster huge datasets that do not fit in memory.
> Scikit-Learn implements this algorithm in the `MiniBatchKMeans` class.
> You can just use this class like the `KMeans` class:

```
from sklearn.cluster import MiniBatchKMeans

minibatch_kmeans = MiniBatchKMeans(n_clusters=5)
minibatch_kmeans.fit(X)
```

Mini-batch K-Means

> Although the Mini-batch K-Means algorithm is much faster than the regular KMeans algorithm
> Its inertia is generally slightly worse, especially as the number of clusters increases

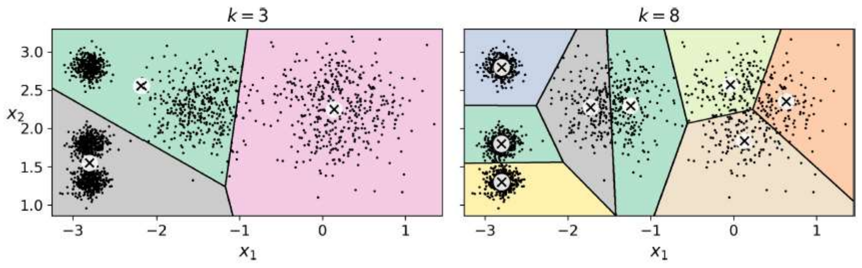


Finding the Optimal Number of Clusters

> So far, we have set the number of clusters *k* to 5 because it was obvious by looking at the data that this is the correct number of clusters.

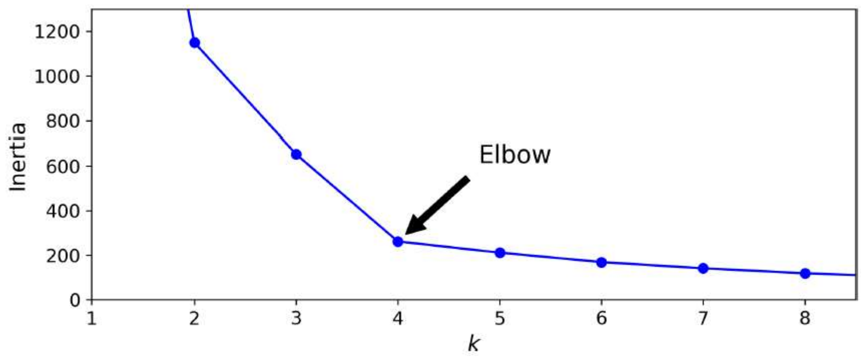
> But in general, it will not be so easy to know how to set *k*

> The result might be quite bad if you set it to the wrong value.



Finding the Optimal Number of Clusters

> You might be thinking that we could just pick the model with the lowest inertia



> The inertia drops very quickly as we increase *k* up to 4

> Then it decreases much more slowly as we keep increasing *k*

Finding the Optimal Number of Clusters

> A more precise approach (also more computationally expensive) is to use the *silhouette score*

> An instance's silhouette coefficient is equal to ($b – a$) / max(a, b)

– 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)

Finding the Optimal Number of Clusters

> 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

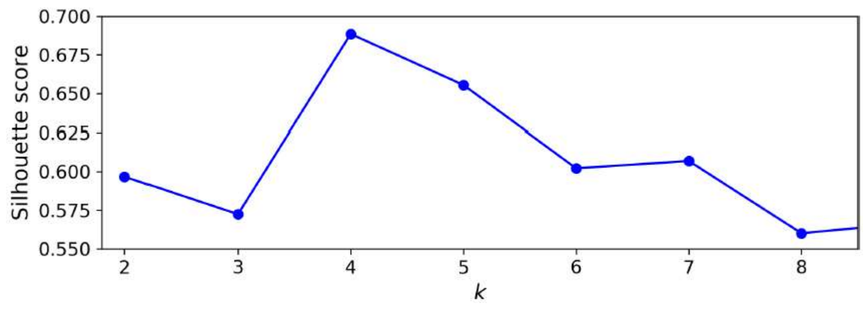
– A coefficient close to -1 means that the instance may have been assigned to the wrong cluster.

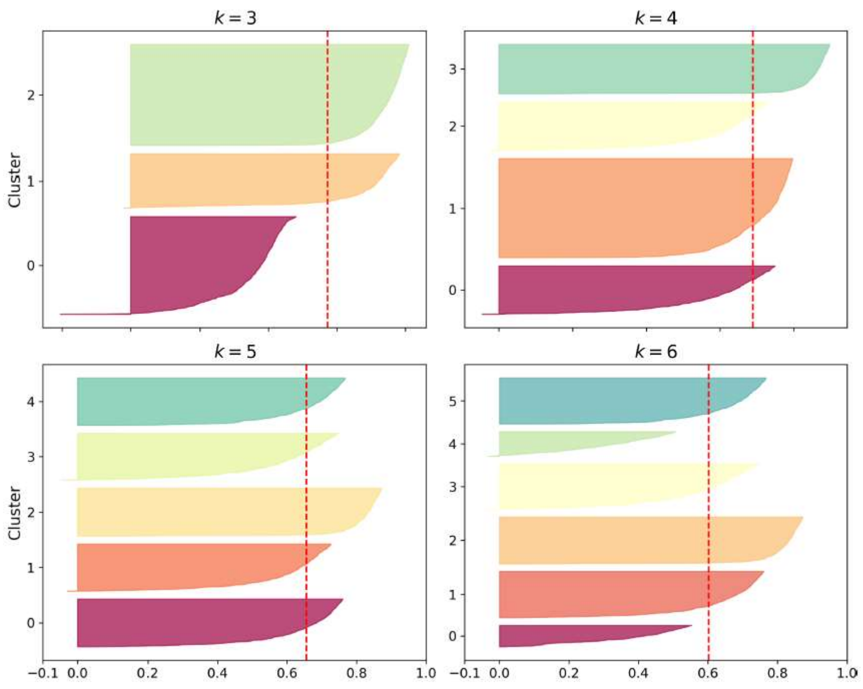
Finding the Optimal Number of Clusters

> To compute the silhouette score, you can use Scikit-Learn's **`silhouette_score()`** function

– Gives all the instances in the dataset, and the labels they were assigned:

```
>>> from sklearn.metrics import silhouette_score
>>> silhouette_score(X, kmeans.labels_)
0.655517642572828
```





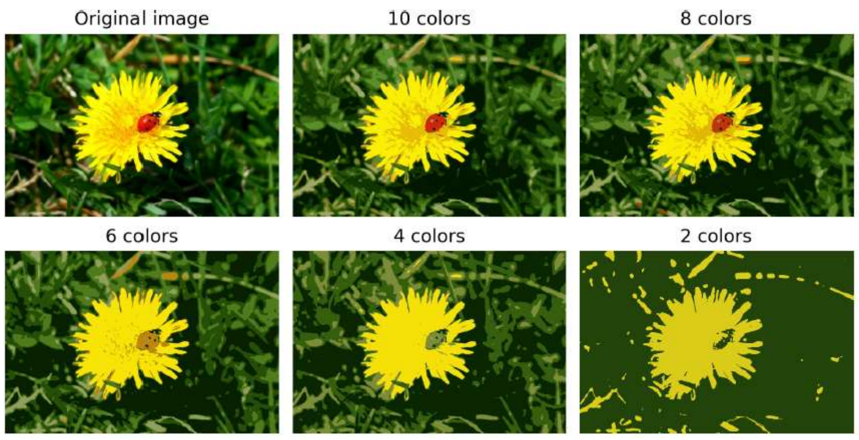
Using clustering for image segmentation

> *Image segmentation* is the task of partitioning an image into multiple segments.

> In *semantic segmentation*, all pixels that are part of the same object type get assigned to the same segment.

```
X = image.reshape(-1, 3)
kmeans = KMeans(n_clusters=8).fit(X)
segmented_img = kmeans.cluster_centers_[kmeans.labels_]
segmented_img = segmented_img.reshape(image.shape)
```

Using clustering for image segmentation



Using Clustering for Semi-Supervised Learning

> Another use case for clustering is in semi-supervised learning, when we have plenty of unlabeled instances and very few labeled instances

> Let’s train a logistic regression model on a sample of 50 labeled instances from the digits dataset:

```
n_labeled = 50
log_reg = LogisticRegression()
log_reg.fit(X_train[:n_labeled], y_train[:n_labeled])
```

> What is the performance of this model on the test set?

```
>>> log_reg.score(X_test, y_test)
0.8266666666666667
```