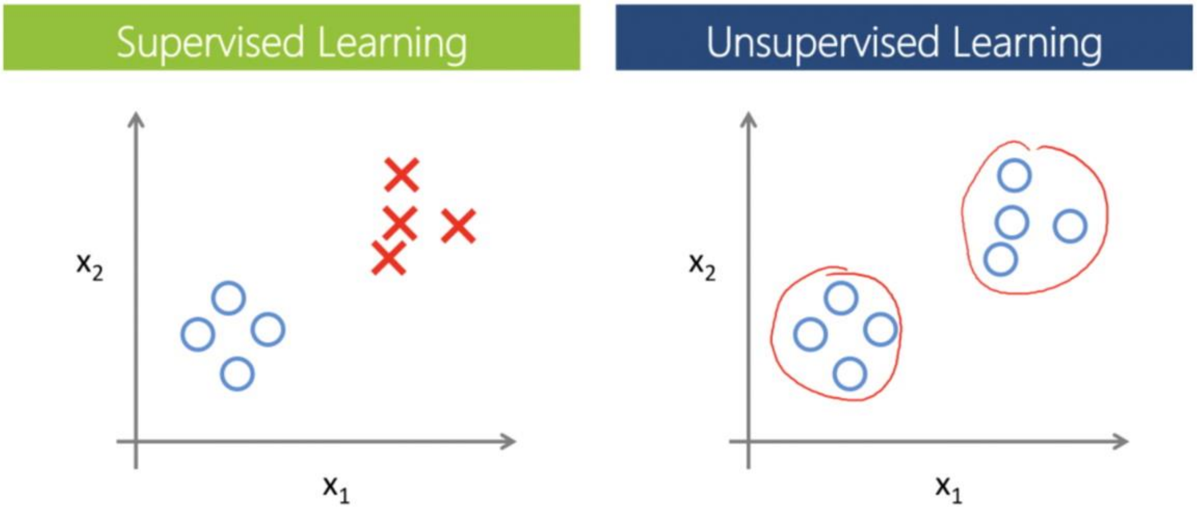
Unsupervised Learning Algorithms cheat-sheet

by Dmytro Nikolaiev

Contents

Introduction

Unsupervised learning is a machine learning technique in which developers don't need to supervise the model. Instead, it allows the model to work on its own to discover patterns and information that was previously undetected. It mainly deals with the unlabeled data, while supervised learning, as we remember, deals with labeled data.



Supervised vs Unsupervised Learning. [Public Domain]

The most popular unsupervised learning tasks are:

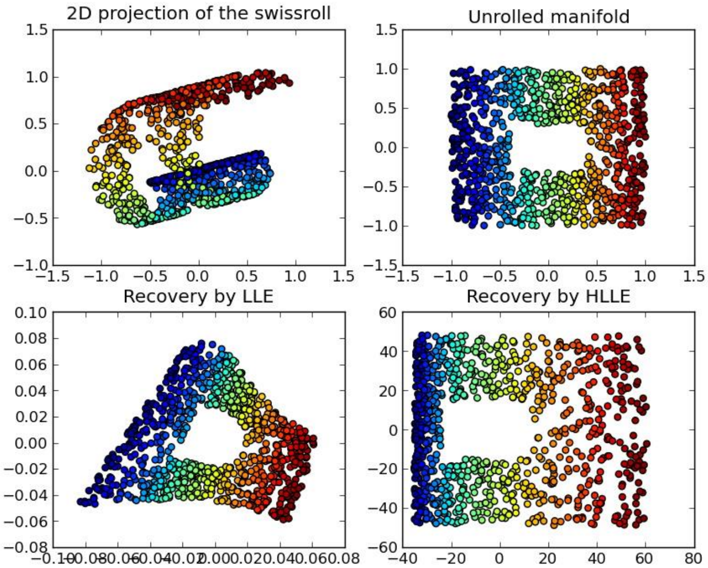
- **Dimensionality Reduction** - task of reducing the number of input features in a dataset,
- **Anomaly Detection** - the task of detecting instances that are very different from the norm,
- **Clustering** - task of grouping similar instances into clusters.

The *Clustering* task is probably the most important in unsupervised learning, since it has many applications. At the same time, *Dimensionality Reduction* and *Anomaly Detection* tasks can be attributed to auxiliary ones (they are often interpreted as *data cleaning* or *feature engineering* tools). Despite the fact that these tasks are definitely important, some people often do not distinguish them separately when studying unsupervised learning, leaving only the clustering task.

Dimensionality Reduction

Dimensionality reduction refers to techniques that *reduce the number of input variables* in a dataset. More input features often make a predictive modeling task more challenging to model, what is known as **the Curse of Dimensionality**.

In the example below the task is to reduce the number of input features (unroll swissroll from 3D to 2D) and save the largest ratio of information at the same time. This is the essence of the dimensionality reduction task and these algorithms.



Dimensionality Reduction Example. [Public Domain]

High-dimensionality statistics and dimensionality reduction techniques are often used for *data visualization*. Nevertheless, these techniques can be used in applied machine learning to simplify a classification or regression dataset in order to better fit a predictive model.

Methods are commonly divided into:

- **Feature Selection** - find a subset of the input features
- **Feature Projection** (or *Feature Extraction*) - find optimal projection of the original data into some low-dimensional space

Next we will talk about the second group of methods. Check *feature engineering* for more *feature selection* tools, e.g. LASSO regression, correlation analysis etc. In fact, the following algorithms can also be used as *feature selection* tools with the difference that these will no longer be the original features, but some of their modifications (for example linear combinations in case of *PCA*.)

Principal Component Analysis

To reduce the dimensionality, Principal Component Analysis (*PCA*) uses the projection of the original data into the *principal components*. The principal components are orthogonal vectors that describe the maximum amount of residual variation (they are found using *Singular Value Decomposition (SVD)*).

Thus, by choosing the first `N` principal components (where `N < M, M is the number of features`), we move from the M-dimensional space to the N-dimensional space, where new features are linear combinations of the existing features.

To select the number of components, the so-called *elbow method* is used. Plot a graph of the cumulative sum of the explained variance and then select the number of components that explains the desired ratio of information (usually 80% or 95%).

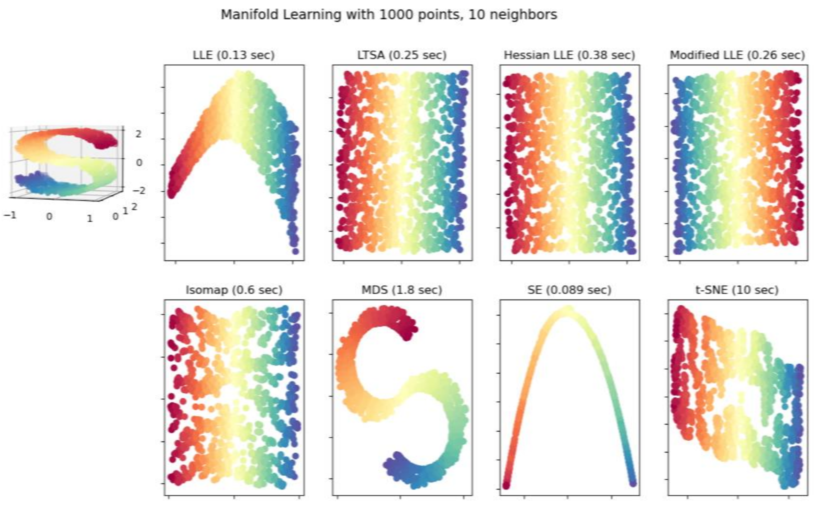
PCA requires data scaling and centering (`sklearn.decomposition.PCA` class does it automatically).

There are a lot of popular modifications of this algorithms, but the most popular are:

- *Incremental PCA* - for *online learning* or when data doesn't fit in memory
- *Randomized PCA* - stochastic algorithm that allows to quickly estimate the first N components
- *Kernel PCA* - *kernel trick* allows to perform complex nonlinear projections

Manifold Learning

Manifold Learning algorithms are based in some distance measure conservation. These algorithms are reducing the dimensionality *while saving distances between objects*.



Comparison of Manifold Learning methods by Scikit Learn. [Image Source]()

- **LLE**

  LLE (*Locally Linear Embedding*) studies the linear connections between data points in the original space, and then tries to move to a smaller dimensional space, while preserving within local neighborhoods. There are a lot of modifications of this algorithm, like *Modified Locally Linear Embedding (MLLE), Hessian-based LLE (HLLE)* and others.

- **Isomap**

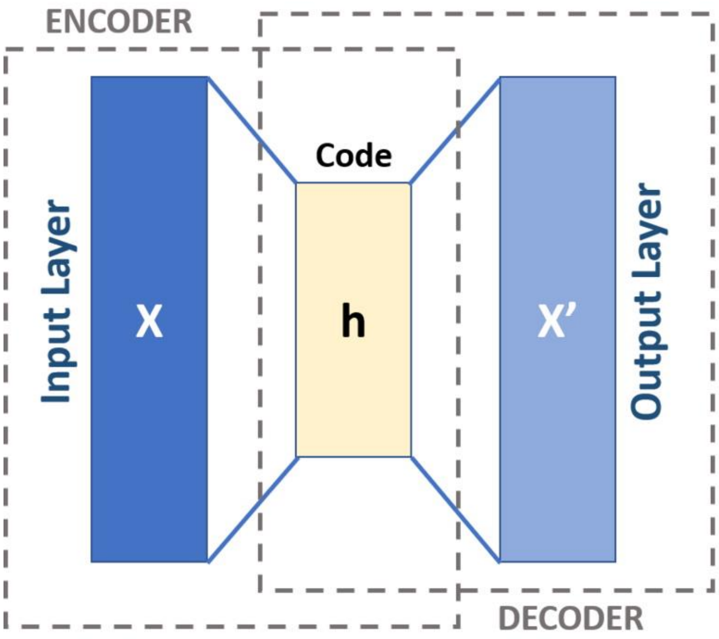
  Isomap (short for *Isometric Mapping*) creates a graph by connecting each instance to its nearest neighbors, and then reduces dimensionality while trying to preserve the *geodesic distances* (distance between two vertices in a graph) between the instances.

- **t-SNE**

  t-SNE stands for *t-distributed Stochastic Neighbor Embedding*. Reduces dimensionality with saving the relative distance between points in space - so it keeps similar instances close to each other and dissimilar instances apart. Most often used for data visualization.

Autoencoders

We can use neural networks to do dimensionality reduction too. Autoencoder is a network that tries to output values that are as similar as possible to the inputs, when the network structure implies *a bottleneck* - a layer where the number of neurons is much fewer than in the input layer.



Autoencoder Structure. [Public Domain]()

If we use a linear activation function, we will get linear dimensionality reduction rules, like *PCA*. But if we use nonlinear activation functions, we can get more complex latent representations. Unfortunately, we have to have a lot of data. Fortunately, this data is unlabeled, so it's usually easy to collect.

As for other algorithms, there are a lot of different variations, like:

- *Denoising Autoencoders* that can help clean up the images or sound
- *Variational Autoencoders*. The latent space is in this case composed by a mixture of distributions instead of a fixed vector
- *Concolutional Autoencoders* for images
- *Reccurent Autoencoders* for time series or text

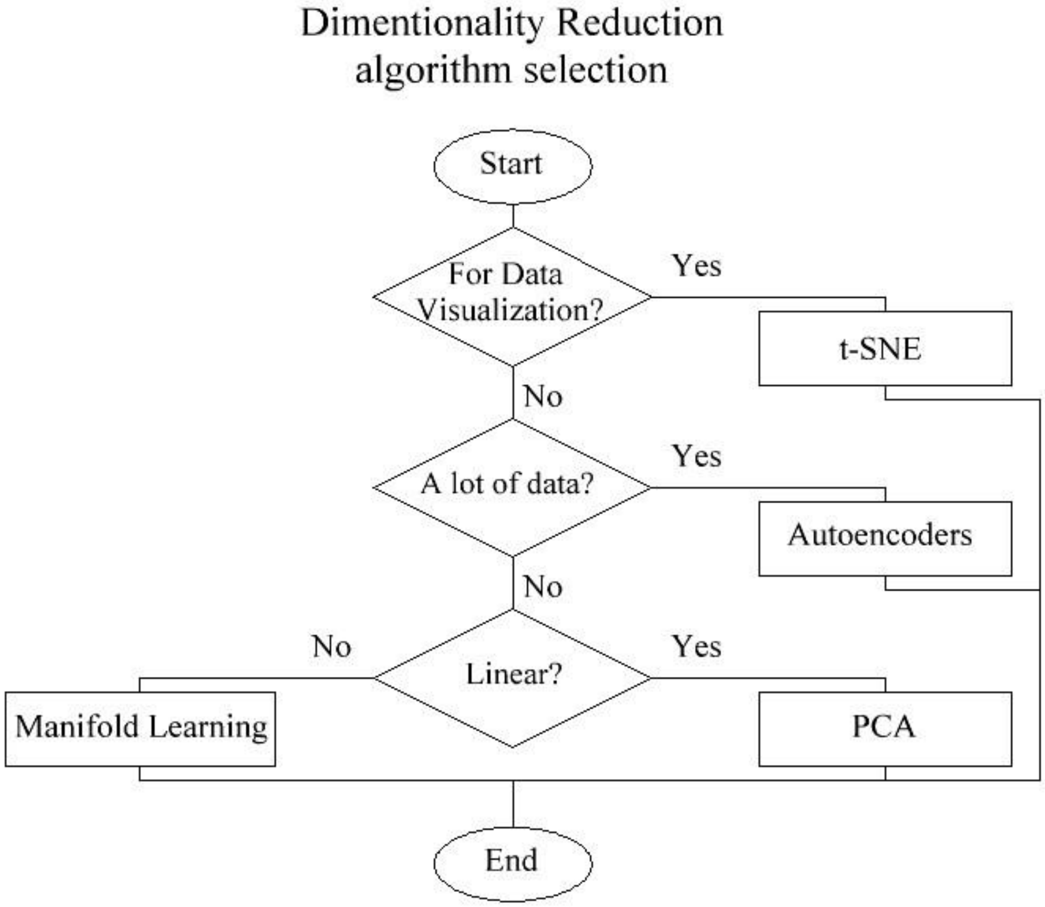
How to choose a dimensionality reduction algorithm?

First of all make sure you scaled the data. Almost all dimensionality reduction algorithms require that.

If you reduce dimensionality for *data visualization*, you should try **t-SNE** first.

If you have a lot of data, **autoencoders** can help you to find very complex latent representations.

If you don't have a lot of data, try **PCA for linear** dimensionality reduction and **manifold learning algorithms** (*LLE*, *Isomap* and others) **for non-linear dimensionality reduction**.



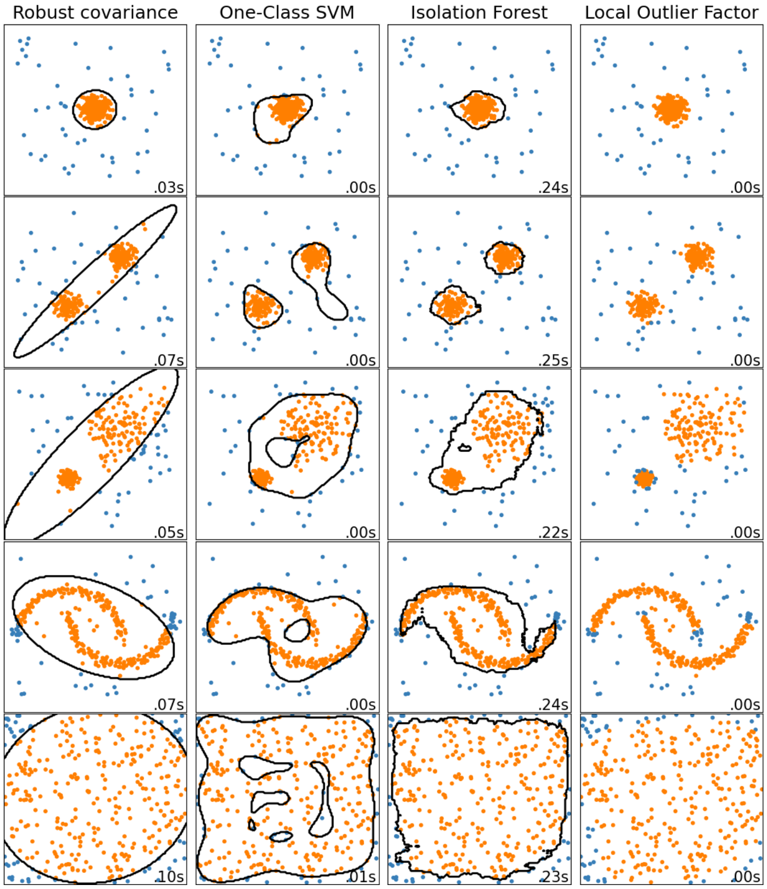
Dimensionality Reduction Algorithm Selection. Image by Author

Note, that almost every algorithm has many variations, and there are a lot of others less popular algorithms, like:

- *Non-negative matrix factorization (NMF)*
- *Random Projections*
- *Linear Discriminant Analysis (LDA)*
- *Multidimensional Scaling (MDS)*
- and others

Anomaly Detection

Anomaly detection (also called **outlier detection**) is the task of detecting instances that are very different from the norm. These instances are called *anomalies*, or *outliers*, while the normal instances are called *inliers*.



Anomaly Detection Algorithms by Scikit Learn. Image Source

Anomaly detection is useful in a wide variety of applications, such as fraud detection, detecting defective products in manufacturing or *data cleaning* - removing outliers from a dataset before training another model.

A closely related task is **Novelty Detection**. It differs from anomaly detection in that the algorithm is assumed to be trained on a clean dataset (not containing outliers), whereas anomaly detection does not make this assumption. It is widely used in *online learning* when it is necessary to determine whether a new entity is an outlier or not.

Another related task is **Density Estimation** - task of estimating the *probability density function* of the random process that generated the dataset. Density estimation is commonly used for anomaly detection (instances located in low-density regions are likely to be anomalies) and data analysis. It is usually solved with *Gaussian Mixture Models*.

Isolation Forest

The algorithm builds a *Random Forest* in which each Decision Tree is grown randomly. The dataset gradually gets chopped into pieces this way, until all instances end up isolated from other instances. Anomalies are usually far from other instances, so on average they tend to get isolated in fewer steps than normal instances.

Local Outlier Factor

Local Outlier Factor (*LOF*) algorithm compares the density of instances around a given instance to the density around its neighbors. An anomaly is often more isolated than its k nearest neighbors.

Minimum Covariance Determinant

Minimum Covariance Determinant (*MCD* or *Fast-MCD*) is useful for outlier detection, in particular for data cleaning. It assumes that:

- the normal instances are generated from a single Gaussian distribution, and
- the dataset is contaminated with outliers that were not generated from this Gaussian distribution.

It is implemented in the `EllipticEnvelope` class.

Some Dimensionality Reduction Algorithms

We can use any algorithm that solves density estimation task, like *Gaussian Mixture Models* and *DBSCAN*. This is a simple and often quite efficient anomaly detection approach. Any instance located in a low-density region can be considered an anomaly, you just need to set some density threshold.

Also, any other algorithms that have the `inverse_transform()` method can be used. This is due to the fact that the *reconstruction error* of an anomaly is always much larger than the one of a normal instance.

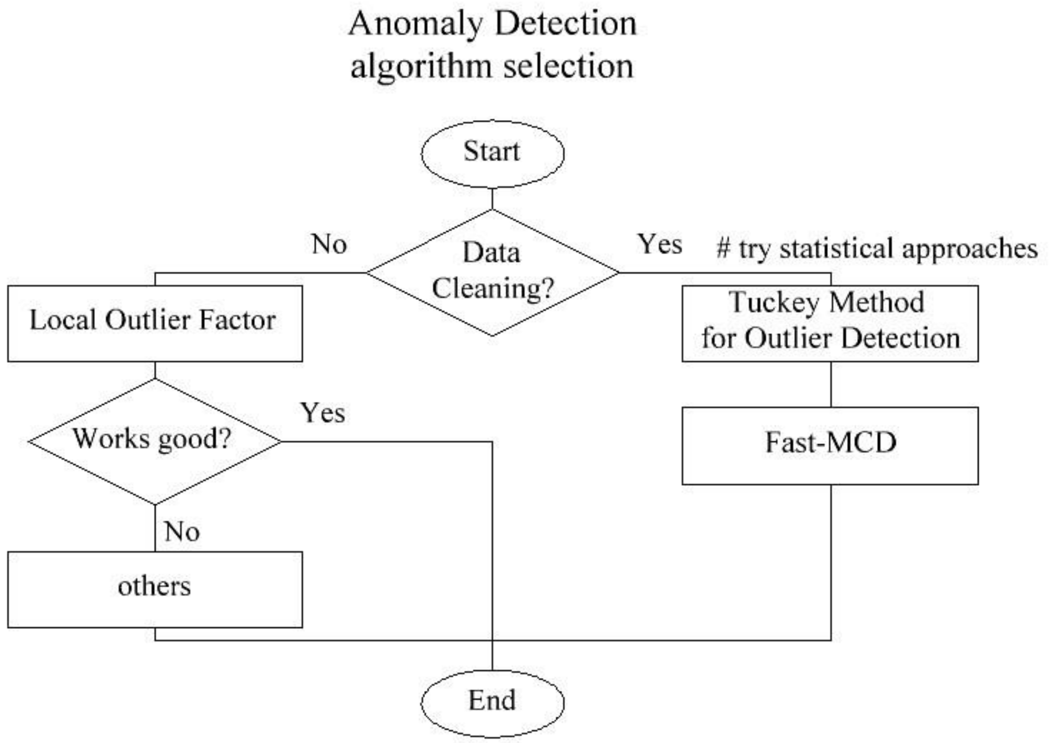
One-class SVM or *kernelized SVM* is also widely used for anomaly detection, but it is better suited for novelty detection. It works great, especially with high-dimensional datasets, but like all SVMs it does not scale to large datasets.

How to choose an anomaly detection algorithm?

If you need to clean up the dataset, you should first try classic statistical methods like **Interquartile Range (IQR)** or **Tuckey Method for Outlier Detection**.

Then try simple and fast **LOF**, and if it doesn't work well - other algorithms according to your task and dataset:

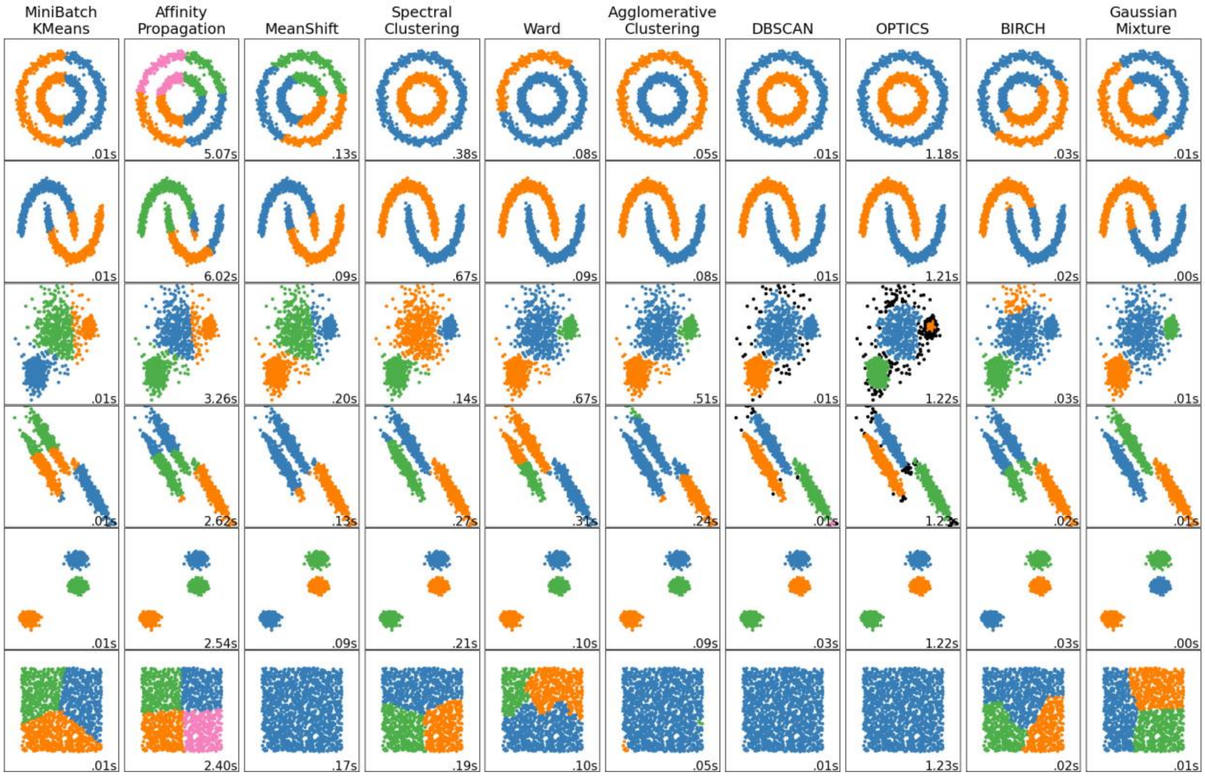
- **One-class SVM** for high-dimensional datasets or *novelty detection*
- **Gaussian Mixture Models** if you can assume that data were generated from a mixture of several Gaussian distributions
- and so on.



Anomaly Detection Algorithm Selection. Image by Author

Clustering

Clustering is the task of dividing the population of unlabeled data points into a number of groups in such a way that objects in the same group (called *a cluster*) are more similar to each other than to those in other clusters.



Clustering Algorithms by Scikit Learn. Image Source

Clustering is used in a wide variety of applications, including these:

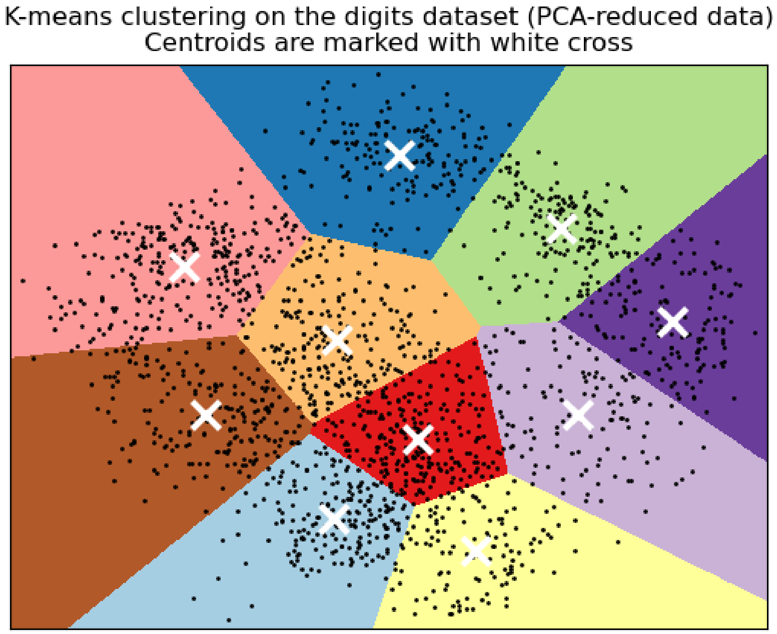
- Semi-supervised learning
- Data analysis - when analyzing a new dataset it can be helpful to run a clustering algorithm, and then analyze each cluster separately
- For anomaly detection - instance, that doesn't belong to any cluster can be considered an anomaly
- Customer segmentation, recommender systems, search engines, image segmentation etc.

All clustering algorithms **requires data preprocessing and standardization**. Most clustering algorithms perform worse with a large number of features, so it is sometimes recommended to use methods of *dimensionality reduction* before clustering.

K-Means

Algorithm starts with random (or not random) *centroids* initializations, which are used as the beginning points for every cluster. After that, we iteratively do the following:

- Assign the observations a cluster number with the nearest center, and
- Move the cluster centroids to the new average value of the cluster elements.



K-Means Clustering by Scikit Learn. ***Image Source***

To choose a good number of clusters we can use *sum of squared distances from points to cluster centroids* as metric and choose the number when this metric stopped falling fast. Other metrics that can be used - *inertia* or *silhouette*.

Speeded version of this algorithm is **Mini Batch K-Means**, when we use a random subsample of the dataset instead of the whole dataset for calculations. There are a lot of other modifications, many of which are implemented in *sklearn*.

Pros:

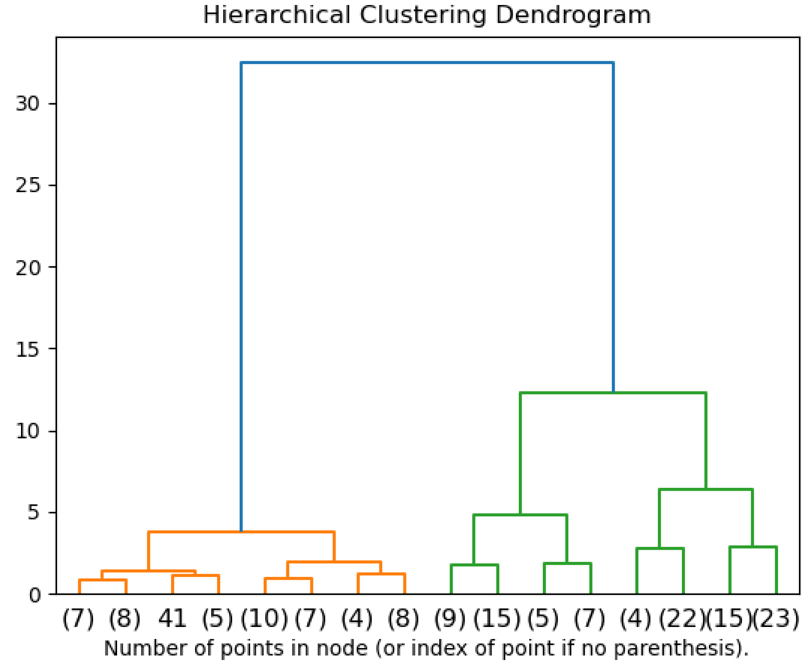
- Simple and intuitive
- Scales to large datasets
- As a result of the algorithm, we have a centroid that can be used as a standard representative of the cluster

Cons:

- The number of clusters must be specified
- Does not cope well with a very large number of features
- Separates only convex and homogeneous clusters well
- Can end up covering to poor solutions, so it needs to be run several times, keeping only the best solution (`n_init` parameter in *sklearn*)

Hierarchical Clustering

Hierarchical clustering (sometimes **Hierarchical Cluster Analysis (HCA)** or **Agglomerative Clustering**) is a general family of clustering algorithms that build nested clusters by merging or splitting them successively. This hierarchy of clusters is represented as a tree (or dendrogram). The root of the tree is the unique cluster that gathers all the samples, and the leaves are the clusters with only one sample.



Hierarchical Clustering Dendrogram Example by Scikit Learn. Image Source

According to the generated dendrogram, you can choose desired separation into any number of clusters. This family of algorithms requires calculating the distance between clusters. For this purpose, different metrics are used, one of the most popular is **Ward distance**.

Pros:

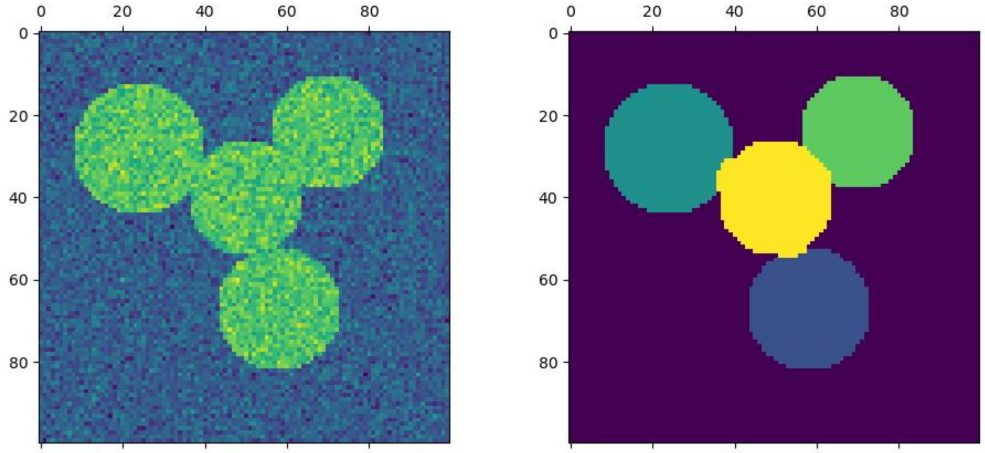
- Simple and intuitive
- Works well when data has a hierarchical structure

Cons:

- The number of clusters must be specified
- Greedy algorithm
- Separates only convex and homogeneous clusters well

Spectral Clustering

This algorithm takes a similarity matrix between the instances and creates a low-dimensional embedding from it (i.e., reduces its dimensionality), and then uses another clustering algorithm in this low-dimensional space (*sklearn* implementation uses K-Means).



Spectral Clustering for image segmentation by Scikit Learn. [Image Source]()

Spectral clustering can capture complex cluster structures and it can also be used to cut graphs (e.g. to identify clusters of friends on a social network).

Pros:

- Can capture complex cluster structures
- Can also be used to cut graphs

Cons:

- Doesn't scale well large numbers of instances
- Doesn't behave well when the clusters have very different sizes

DBSCAN

DBSCAN stands for *Density-Based Spatial Clustering of Applications with Noise*.

The DBSCAN algorithm views clusters as areas of high density separated by areas of low density. The central component to the DBSCAN is the concept of *core samples*, which are samples that are in areas of high density. A cluster is therefore a set of core samples, each close to each other and a set of non-core samples that are close to a core sample. Other samples are defined as outliers (or anomalies).

An extension or generalization of the DBSCAN algorithm is the **OPTICS** algorithm (*Ordering Points To Identify the Clustering Structure*).

Pros:

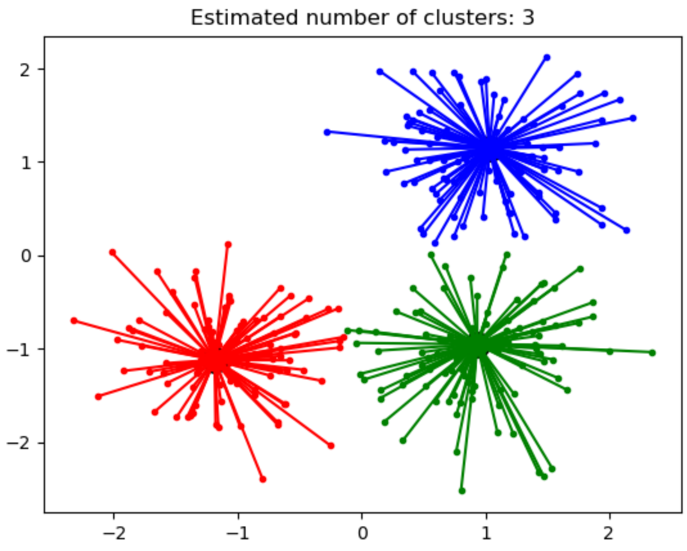
- You don't need to specify the number of clusters
- Solves the anomaly detection task at the same time

Cons:

- Need to select the density parameter
- Does not cope well with a sparse data

Affinity Propagation

Affinity Propagation creates clusters by sending messages between pairs of samples until convergence. A dataset is then described using a small number of examples (standard representatives), which are identified as those most representative of other samples.



Affinity Propagation Clustering by Scikit Learn. Image Source

Unfortunately, this algorithm has a computational complexity of O(m^2), so it too is not suited for large datasets.

Pros:

- You don't need to specify the number of clusters
- As a result of the algorithm we have standard representatives of a cluster. Unlike K-Means, these representatives are not just mean values, but real objects from the train set.

Cons:

- Computational complexity of O(m^2), so it too is not suited for large datasets
- Separates only convex and homogeneous clusters well
- Usually works worse than other algorithms

Mean Shift

This algorithm starts by placing a circle centered on each instance, then for each circle it computes the mean of all the instances located within it, and it shifts the circle so that it is centered on the mean. Next, it iterates this mean-shifting step until all the circles stop moving.

Mean Shift shifts the circles in the direction of higher density, until each of them has found a local density minimum. Finally, all the instances whose circles have settled in the same place (or close enough) are assigned to the same cluster. Mean Shift has some of the features of DBSCAN, because it's based on density too.

Pros:

- You don't need to specify the number of clusters
- Have just one hyperparameter - the radius of the circles, called *bandwidth*

Cons:

- Does not cope well with a sparse data
- Tends to chop clusters into pieces when they have internal density variations
- Computational complexity of O(m^2), so it too is not suited for large datasets

BIRCH

The BIRCH stands for *Balanced Iterative Reducing and Clustering using Hierarchies*. This algorithm was designed specifically for very large datasets, and it can be faster than batch K-Means, with similar results, as long as the number of features is not too large (<20). During training, it builds a tree structure containing just enough information to quickly assign each new instance to a cluster, without having to store all the instances in the tree: this approach allows it to use limited memory, while handling huge datasets.

Pros:

- Was designed specifically for very large datasets when number of features is not too large (<20)
- Allows it to use limited memory, while handling huge datasets

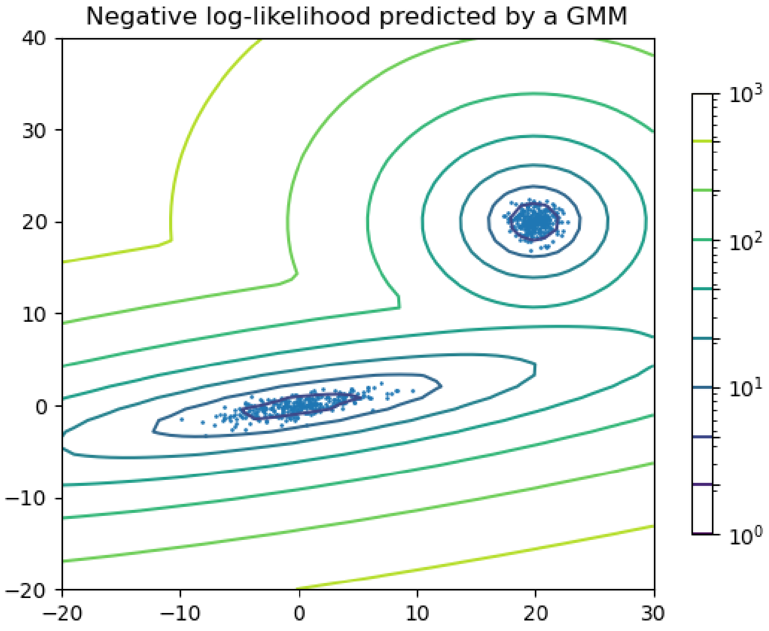
Cons:

- The number of clusters must be specified
- Does not cope well with a high-dimensional data

Gaussian Mixture Models

Gaussian Mixture Models (*GMM*) is a probabilistic model that can solve both *Clustering* and *Anomaly detection*/*Density Estimation* unsupervised learning tasks.

This method relies on the *Expectation Maximization (EM)* algorithm and assumes that the data instances were generated from a mixture of several Gaussian distributions whose parameters are unknown.



Density Estimation with GMM by Scikit Learn. Image Source

To choose a good number of clusters we can use *BIC* (Bayesian information criterion) or *AIC* (Akaike information criterion) and choose the model with minimum value. On the other hand, **Bayesian GMM** can be used - this model can detect the number of clusters itself and requires only a value that is greater than the optimal number of clusters.

The Gaussian Mixture Model is a *generative model*, meaning you can sample new instances from it. It is also possible to estimate the density of the model at any given location.

Pros:

- Perfectly deals with data instances that were generated from a mixture of Gaussian distributions with different shapes and sizes
- At the same time solves *density estimation* task
- Is a *generative model*

Cons:

- The number of clusters must be specified (not in case of *Bayesian GMM*)
- *Expectation Maximization* algorithm can end up covering to poor solutions, so it needs to be run several times, keeping only the best solution (`n_init` parameter in *sklearn*)
- Does not scale well large numbers of features
- Assume that data instances were generated from a mixture of Gaussian distributions, so cope bad with data of other shape

How to choose a clustering algorithm?

As you can see, clustering task is quite difficult and have a wide variety of applications, so it's almost impossible to build some universal set of rules to select a clustering algorithm - all of them have advantages and disadvantages.

Things become better when you have some assumptions about your data, so *data analysis* can help you with that. What is the approximate number of clusters? Are they located far from each other or do they intersect? Are they similar in shape and density? All that information can help you to solve your task better.

If the number of clusters is unknown, a good initial approximation is *the square root of the number of objects*. You can also first run an algorithm that does not require a number of clusters (*DBSCAN* or *Affinity Propagation*) and start from this number.

But the most important question remains the evaluation of quality - you can try all the algorithms, but how to understand which one is the best? There are a great many metrics for this - from *homogeneity* and *completeness* to *silhouette* - they show themselves differently in different tasks.

Conclusions

In this article I tried to describe all main unsupervised learning tasks and algorithms and give you a big picture of dimensionality reduction, anomaly detection and clustering.

I hope that these descriptions and recommendations will help you and motivate you to learn more and go deeper into machine learning.