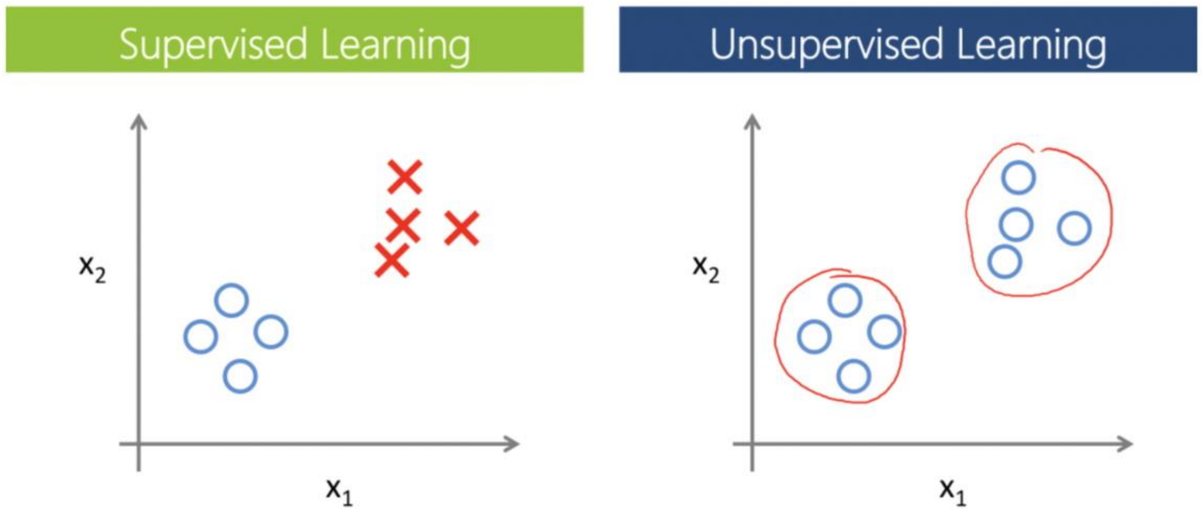
Unsupervised Learning Algorithms cheat sheet

by Dmytro Nikolaiev

Contents

Introduction

Unsupervised learning is a machine learning technique in which developers don't need to supervise the model. Instead, this type of learning allows the model to work independently *without any supervision* to discover hidden 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

Three of the most popular unsupervised learning tasks are:

- **Dimensionality Reduction -** the 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, and
- **Clustering -** the task of grouping similar instances into clusters.

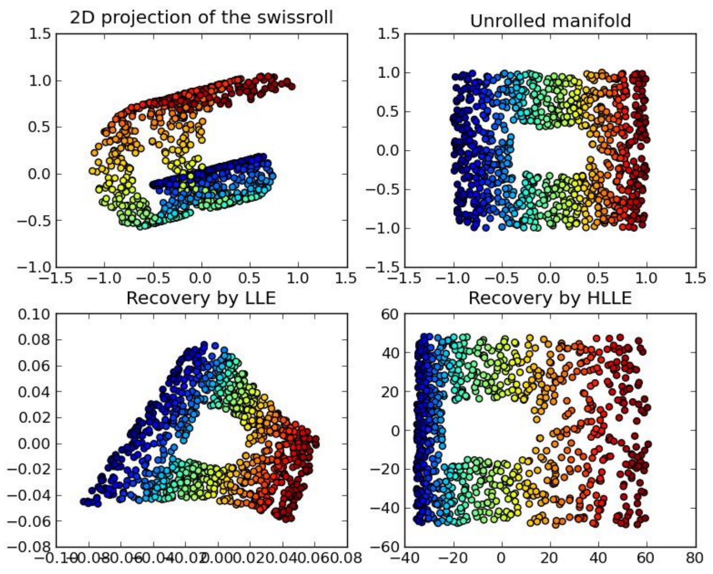
The *Clustering* task is probably the most important in unsupervised learning since it has a wide variety of applications. At the same time, *Dimensionality Reduction* and *Anomaly Detection* tasks can be attributed to auxiliary ones (they are often interpreted as *data cleaning* or *feature engineering* 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.

Each of these three tasks and the algorithms for solving them will be discussed in more detail later in the corresponding sections. However, note that the **Other Unsupervised Learning Tasks** section lists other less popular tasks that can also be attributed to unsupervised learning.

Dimensionality Reduction

Dimensionality reduction algorithms represent techniques that *reduce the number of features* (not samples) in a dataset.

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

Two main applications of dimensionality reduction algorithms are:

- **Data Visualization** & **Data Analysis** - reduce the number of input features to three or two and use data visualization techniques to get insights about the data
- **Preparatory tool for other machine learning algorithms**. More input features often make a prediction task more challenging to model, what is known as the **Curse of Dimensionality**. Since many algorithms (both from supervised and unsupervised learning (e.g. regression/classification, clustering)) do not work well with sparse or high-dimensional data, dimensionality reduction algorithms can greatly increase the quality. Often, this also provides faster and simpler calculations.

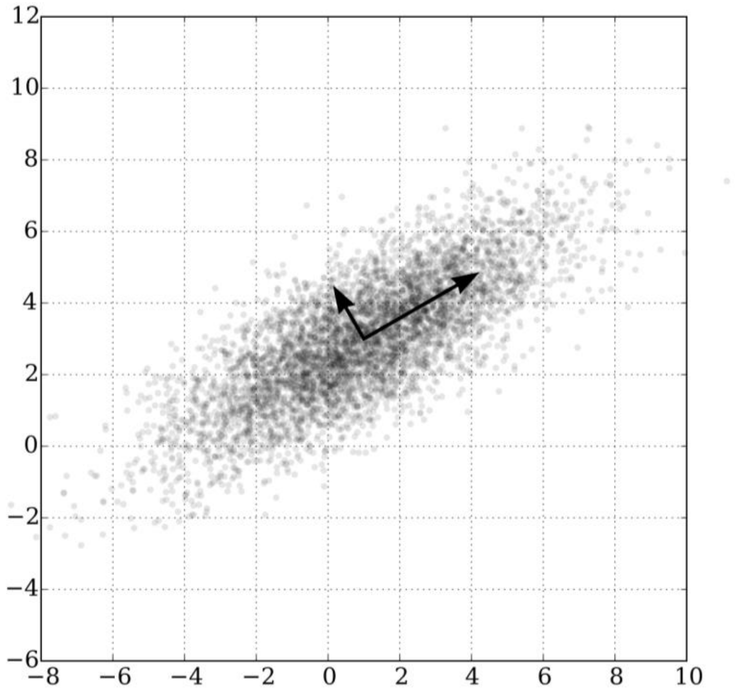
Methods are commonly divided into:

- **Feature Selection** - find a subset of the input features
- **Feature Projection** (or *Feature Extraction*) - find the 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*).



PCA of a Gaussian distribution with two principal components. [Public domain]

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%).

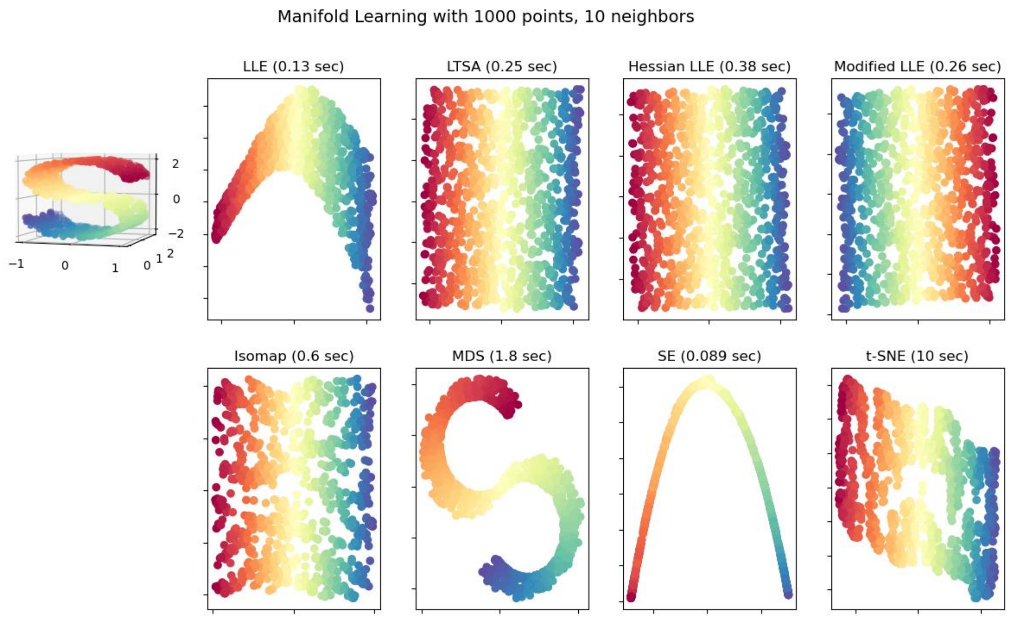
PCA requires data scaling and centering (`sklearn.decomposition.PCA` class does it automatically).

There are a lot of popular modifications of these 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 performing complex nonlinear projections

Manifold Learning

Manifold Learning algorithms are based on 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**

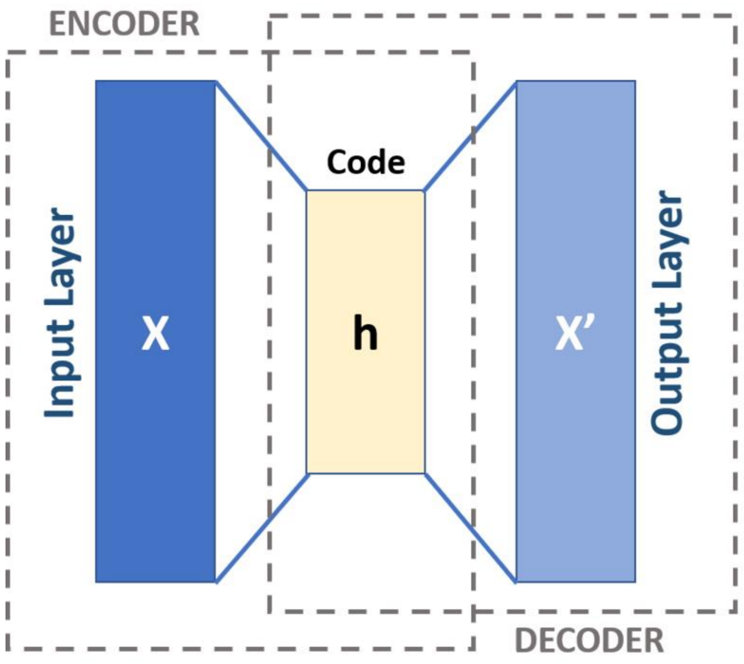
  Isomap (short for Isometric Mapping) creates a graph by connecting each instance to its nearest neighbors, and then reduces dimensionality while trying to preserve the geodesic distances (distance between two vertices in a graph) between the instances.

- **t-SNE**

  t-SNE stands for *t-distributed Stochastic Neighbor Embedding*. Reduces dimensionality by 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* that deal with distributions instead of specific values
- *Convolutional Autoencoders* for images
- *Recurrent Autoencoders* for time series or text

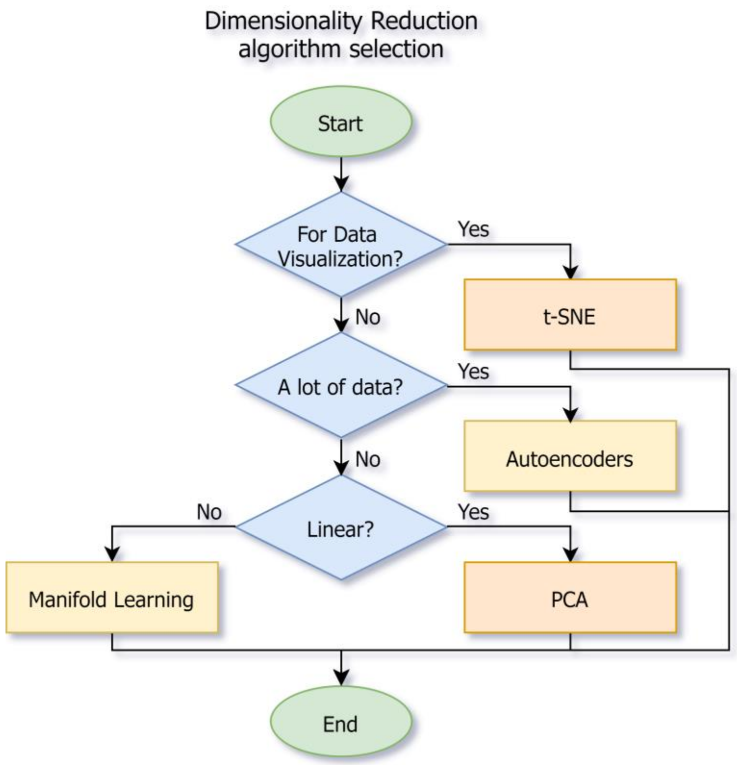
How to choose a dimensionality reduction algorithm?

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**.



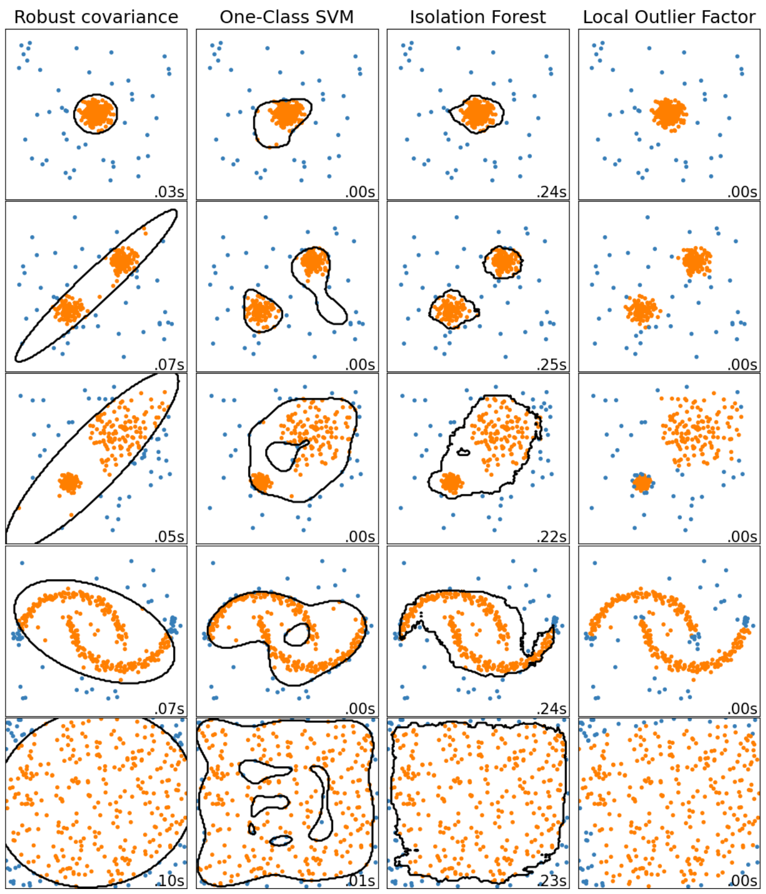
Dimensionality Reduction Algorithm Selection. Image by Author

Note, that almost every algorithm has many variations, and there are a lot of other 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 **outlier detection**) is the task of detecting abnormal instances - instances that are very different from the norm. These instances are called *anomalies*, or *outliers*, while normal instances are called *inliers*.



Anomaly Detection Algorithms by Scikit Learn. [Image source]()

Anomaly detection is useful in a wide variety of applications, the most important are:

- *data cleaning* - removing outliers from a dataset before training another model
- directly *anomaly detection* tasks: fraud detection, detecting defective products in manufacturing etc.

As you may have noticed, some problems of *unbalanced classification* can be also solved using anomaly detection algorithms. But you need to understand the difference - these are *two completely different approaches*.

In the case of **classification, we want to understand what anomalies (positive examples) look like** to detect similar instances in the future. In the case of **anomaly detection, future anomalies may look completely different from any examples we have seen before**. Because our dataset is unlabeled, we can only suspect how normal data points look and consider any other instances as anomalies.

For example, an email spam detection task can be considered as a classification task (when we have enough spam to understand what spam emails should look like) and also as an anomaly detection task (when we have to understand how normal emails look like and consider any other emails as spam).

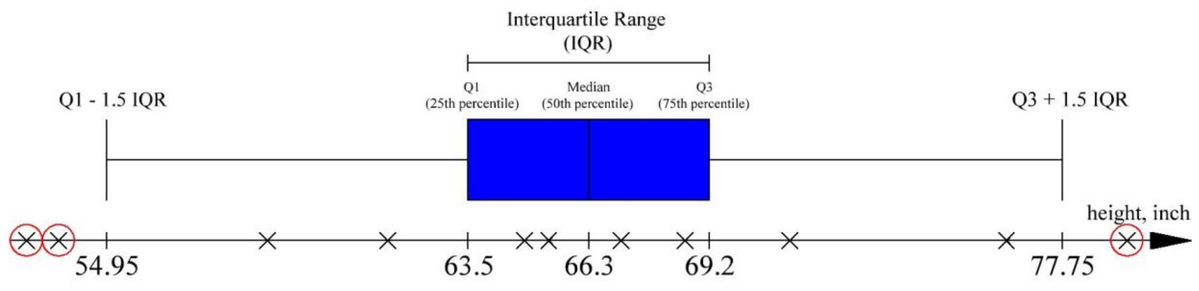
A closely related task is **Novelty Detection**, but in this case, the algorithm is assumed to be trained on a clean dataset (without outliers). It is widely used in *online learning* when it is necessary to identify whether a new instance is an outlier or not.

Another related task is **Density Estimation**. It is a task of estimating the *probability density function* of the random process that the dataset generates. It is usually solved with clustering algorithms, based on the *density* concept (*Gaussian Mixture Models* or *DBSCAN*) and can also help for anomaly detection and data analysis.

Statistical Approaches

The easiest way to detect outliers is to try statistical methods, that were developed a very long time ago. One of the most popular of them is called **Tukey Method for Outlier Detection** (or **Interquartile Range (IQR)**).

Its essence is to calculate percentiles and the interquartile range. Data points located before Q1 - 1.5*IQR and after Q3 + 1.5*IQR are considered outliers. Below you can see an illustration of this method using the people height dataset example. Heights below 54.95 inches (139 cm) and above 77.75 inches (197 cm) are considered outliers.



Tukey Method for Outlier Detection on the example of the heights of people. Image by Author

This and other statistical approaches (*z-score method for detecting outliers*, etc.) are often used for data cleaning.

Clustering and dimensionality reduction algorithms

Another simple, intuitive and often effective approach to anomaly detection is to solve density estimation task with some clustering algorithms, like *Gaussian Mixture Models* and *DBSCAN*. Then, any instance located in regions with a lower density level can be considered an anomaly, we just need to set some density threshold.

Also, any dimensionality reduction algorithm that has the `inverse_transform()` method can be used. This is because the *reconstruction error* of an anomaly is always much larger than the one of a normal instance.

Isolation Forest and SVM

Some supervised learning algorithms also can be used for anomaly detection, and two of the most popular are *Isolation Forest* and *SVM*. These algorithms are better suited for novelty detection but usually work well for anomaly detection too.

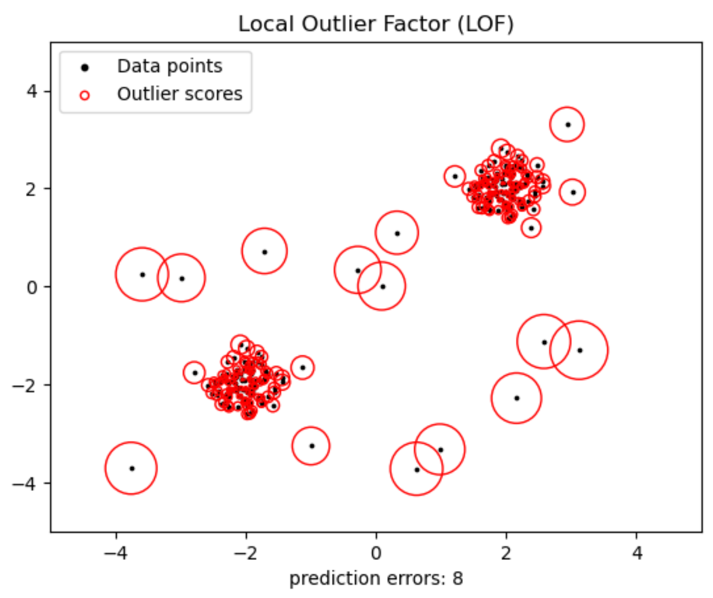
Isolation Forest algorithm builds a *Random Forest* in which each decision tree is grown randomly. With each step, this forest isolates more and more points until all they become isolated. Since the anomalies are located further from the usual data points, they are usually isolated in fewer steps than normal instances. This algorithm performs well for high-dimensional data, but requires a larger dataset than SVM.

SVM (in our case *one-class SVM*) is also widely used for anomaly detection. Thanks to *kernel trick* kernelized SVM can build an effective "limiting hyperplane", which will separate the normal points from the outliers. Like any SVM modification, it copes great with high-dimensional or sparse data, but works well only on small and medium-sized datasets.

Local Outlier Factor

Local Outlier Factor (*LOF*) algorithm is based on the assumption that the anomalies are located in lower-density regions. However, instead of just setting a density threshold (as we can do with *DBSCAN*), it compares the density of a certain point with the density of k its nearest neighbors. If this certain point has a much lower density than its neighbors (that means that it is located far away from them), it is considered an anomaly.

This algorithm can be both used for anomaly and novelty detection. It is used very often because of its computational simplicity and good quality.



Local Outlier Factor by Scikit Learn. Image source

Minimum Covariance Determinant

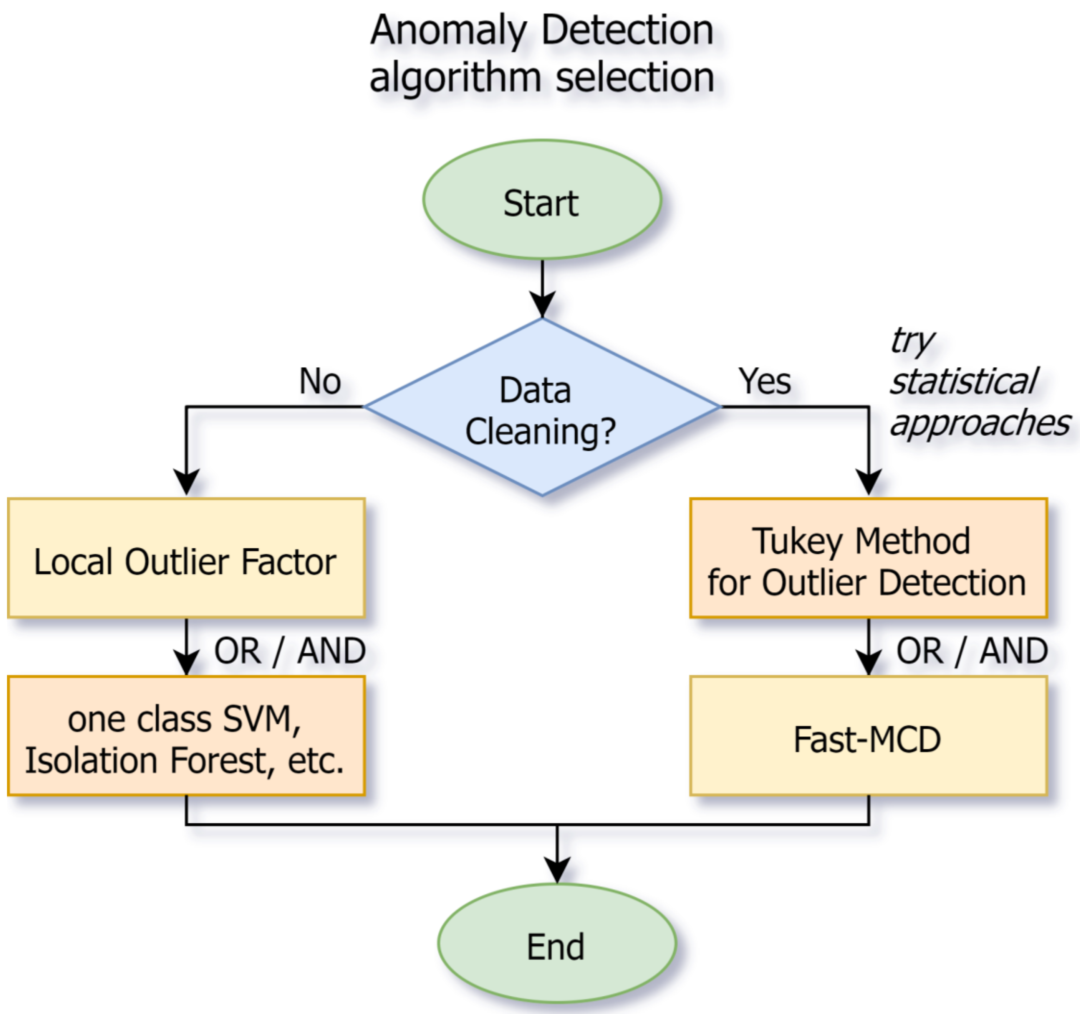
Minimum Covariance Determinant (*MCD* or its modification *Fast-MCD*) is useful for outlier detection, in particular for data cleaning. It assumes that inliers are generated from a single Gaussian distribution, and outliers were not generated from this distribution. Since many data have a normal distribution (or can be reduced to it), this algorithm usually performs well. It is implemented in the `EllipticEnvelope` *sklearn* class.

How to choose an anomaly detection algorithm?

If you need to clean up the dataset, you should first try classic statistical methods like **Tukey Method for Outlier Detection**. Then go with **Fast-MCD**, if you know that your data distribution is Gaussian.

If you do anomaly detection not for data cleaning, first of all, try simple and fast **Local Outlier Factor**. If it doesn't work well (or if you need separating hyperplane for some reason) - try other algorithms according to your task and dataset:

- **One-class SVM** for sparse high-dimensional data or **Isolation Forest** for dense high-dimensional data
- **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
Start
Data Cleaning?
No
Yes
try statistical approaches
Local Outlier Factor
OR / AND
one class SVM, Isolation Forest, etc.
Tukey Method for Outlier Detection
OR / AND
Fast-MCD
End

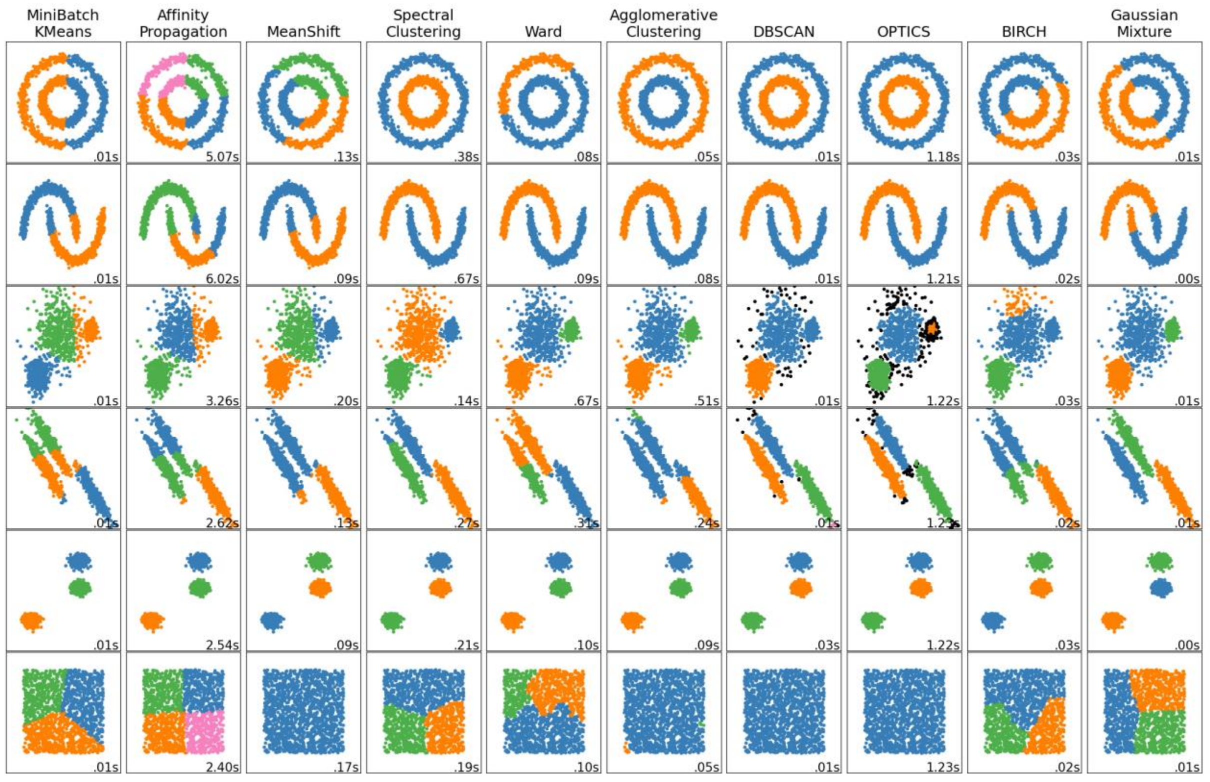
Anomaly Detection Algorithm Selection. Image by Author

Clustering

Clustering (also called *cluster analysis*) is a task of grouping similar instances into clusters. More formally, clustering is the task of grouping the population of **unlabeled data points** into clusters in a way that data points in the same cluster are more similar to each other than to data points in other clusters.

The clustering task is probably the most important in unsupervised learning, since it has many applications, for example:

- *data analysis*: often a huge dataset contains several large clusters, analyzing which separately, you can come to interesting insights;
- *anomaly detection*: as we saw before, data points located in the regions of low density can be considered as anomalies;
- *semi-supervised learning*: clustering approaches often helps you to automatically label partially labeled data for classification tasks;
- *indirectly clustering* tasks (tasks where clustering helps to gain good results): recommender systems, search engines, etc., and
- *directly clustering* tasks: customer segmentation, image segmentation, etc.



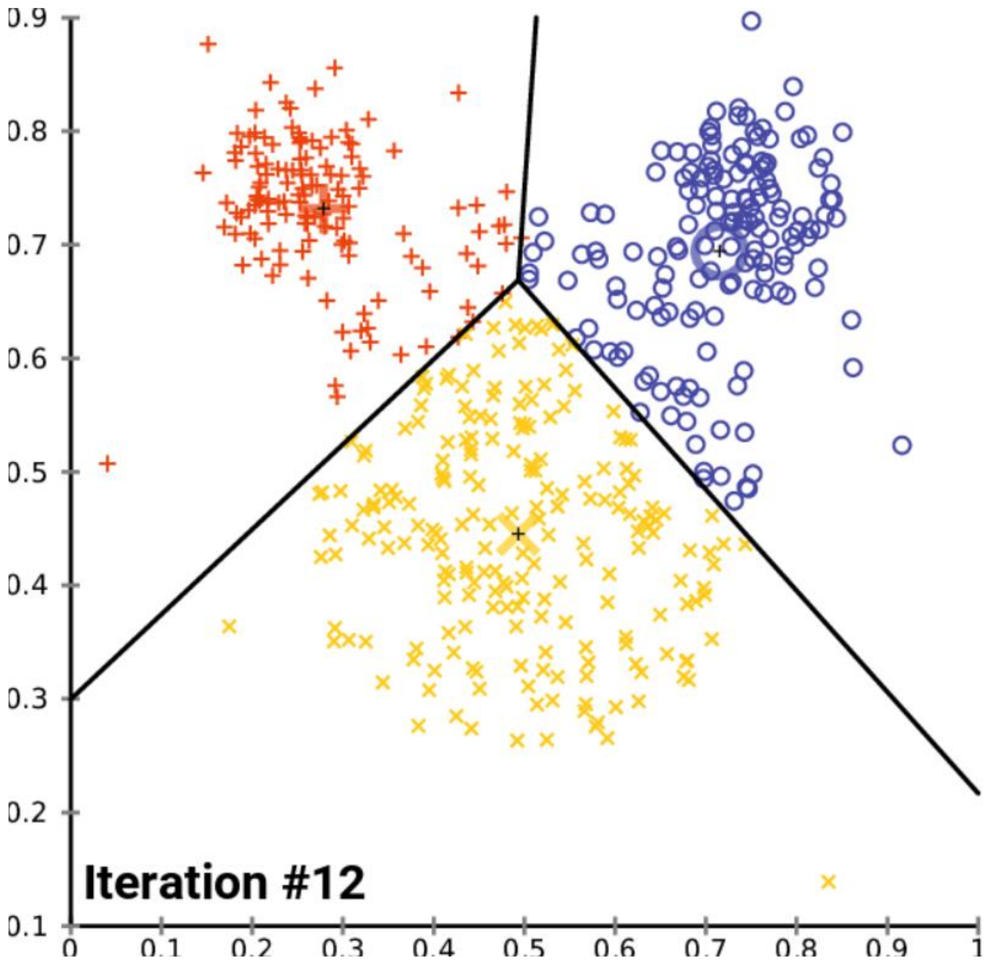
Clustering Algorithms by Scikit Learn. [Image source]()

All clustering algorithms **require 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

K-Means algorithm is based on the *centroid* concept. *Centroid* is a geometric center of a cluster (mean of coordinates of all cluster points). First, centroids are initialized randomly (this is the basic option, but there are other initialization techniques). After that, we iteratively do the two following steps, while centroids are moving:

- *Update the clusters* - for each data point assign it a cluster number with the nearest centroid, and
- *Update the clusters' centroids* - calculate the new mean value of the cluster elements to move centroids.



Convergence of K-Means algorithm. [Public domain]()

The strengths and weaknesses of the algorithm are intuitive. It is simple and fast, but it requires initial knowledge about the number of clusters. It also does not detect clusters of complex shapes well and can result in a local solution. To choose a good number of clusters we can use a *sum of squared distances from data points to cluster centroids* as a metric and choose the number when this metric stops falling fast (elbow method). To find a globally optimal solution, you can run the algorithm several times and choose the best result (`n_init` parameter in *sklearn*).

A speeded version of this algorithm is **Mini Batch K-Means**. In that case, we use a random subsample instead of the whole dataset for calculations. There are a lot of other modifications, and many of them are implemented in *sklearn*.

Pros:

- Simple and intuitive;
- Scales to large datasets;
- As a result, we also have centroids that can be used as standard cluster representatives.

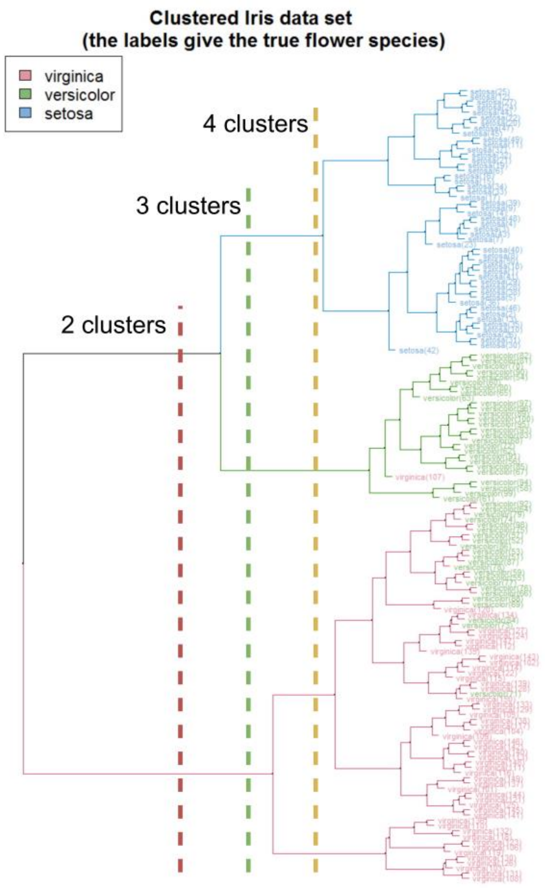
Cons:

- Knowledge about the number of clusters is necessary and must be specified as a parameter;
- Does not cope well with a very large number of features;
- Separates only convex and homogeneous clusters well;
- Can result in poor local solutions, so it needs to be run several times.

Hierarchical Clustering

Hierarchical clustering (also **Hierarchical Cluster Analysis (HCA)** or **Agglomerative Clustering**) is a family of clustering algorithms that build a hierarchy of clusters during the analysis.

It is represented as a dendrogram (a tree diagram). The *root* of the tree (usually the upper or left element) is one large cluster cluster that contains all data points. The *leaves* (bottom or right elements) are tiny clusters, each of which contains only one data point. According to the generated dendrogram, you can choose the desired separation into any number of clusters.



Hierarchical Clustering dendrogram example on the iris dataset.
Adapted from a [public domain](#)

This family of algorithms requires calculating the distance between clusters. Different metrics are used for this purpose (simple linkage, complete linkage, average linkage, centroid linkage, etc.), but one of the most effective and popular is **Ward's distance** or Ward's linkage.

To learn more about the different methods of measuring the distance between clusters, start with [this article](#).

Pros:

- Simple and intuitive;
- Works well when data has a hierarchical structure;
- Knowledge about the number of clusters is not necessary.

Cons:

- Requires additional analysis to choose the resulting number of clusters;
- Separates only convex and homogeneous clusters well;
- A greedy algorithm can result in poor local solutions.

Spectral Clustering

The spectral clustering approach is based on graph theory and linear algebra. This algorithm uses the *spectrum* (set of *eigenvalues*) of the *similarity matrix* (that contains the similarity of each pair of data points) to perform dimensionality reduction. Then it uses some of the clustering algorithms in this low-dimensional space (`sklearn.cluster.SpectralClustering` class uses K-Means).

Due to the dimensionality reduction, this algorithm can detect complex cluster structures and shapes. It can also be used to search for clusters in graphs. However, due to its computational complexity, it does not work well with large datasets.

Pros:

- Can detect complex cluster structures and shapes;
- Can be used to search for clusters in graphs.

Cons:

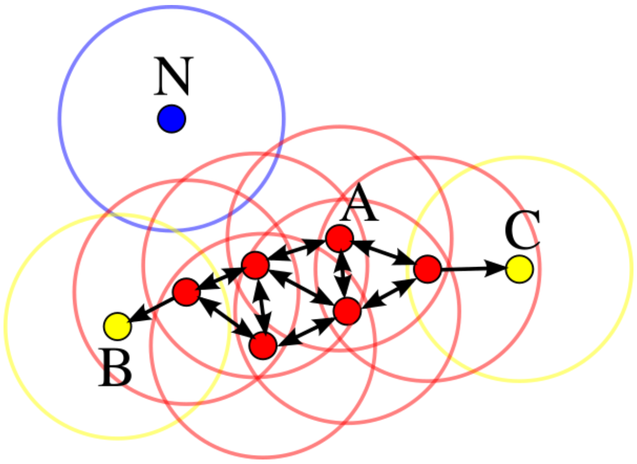
- Knowledge about the number of clusters is necessary and must be specified as a parameter;
- Does not cope well with a very large number of instances;
- Does not cope well when the clusters have very different sizes.

DBSCAN

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

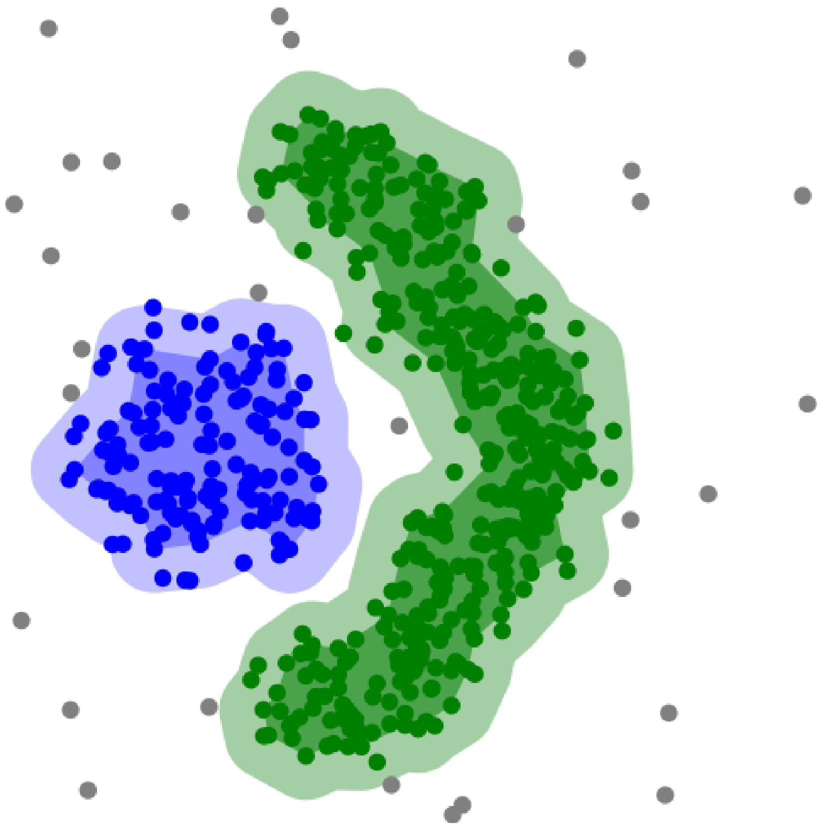
According to this algorithm, clusters are high-density regions (where the data points are located close to each other) separated by low-density regions (where the data points are located far from each other).

The central concept of the DBSCAN algorithm is the idea of a *core sample*, which means a sample located in an area of high density. Data point A is considered a core sample if at least `min_samples` other instances (usually including A) are located within `eps` distance from A.



DBSCAN core samples example. min_samples=4, core samples marked in red. [Public domain](#)

Therefore, a cluster is a group of core samples located close to each other and some non-core samples located close to core samples. Other samples are defined as outliers (or anomalies) and do not belong to any cluster. This approach is called **density-based clustering**. It allows you not to specify the number of clusters as a parameter and find clusters of complex shapes.



DBSCAN clustering. [Public domain](#)

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

Pros:

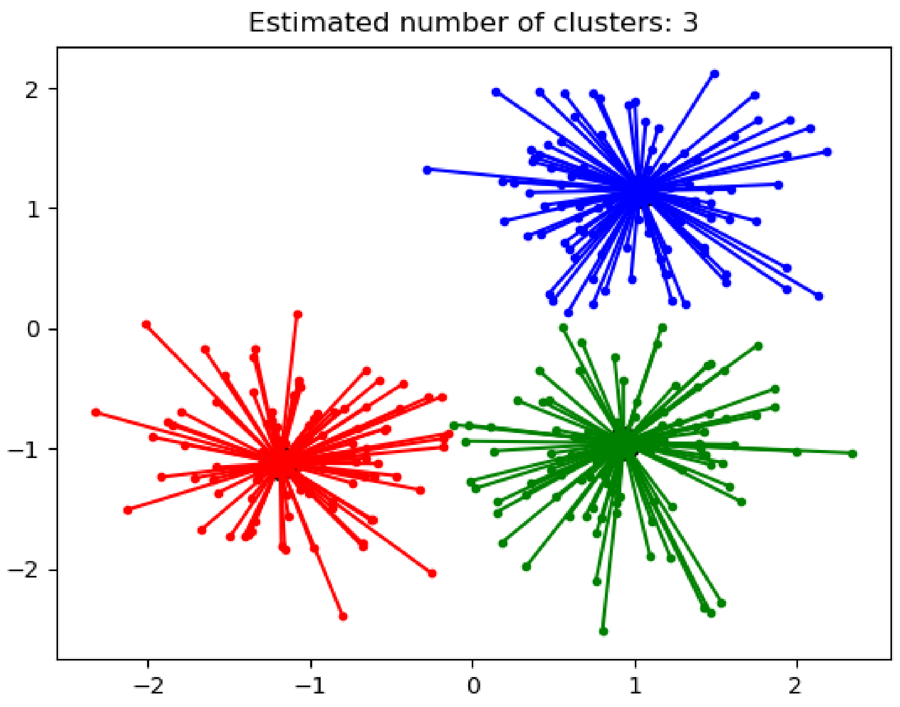
- Knowledge about the number of clusters is not necessary;
- Also solves the *anomaly detection* task.

Cons:

- Need to select and tune the density parameter (`eps`);
- Does not cope well with sparse data.

Affinity Propagation

The Affinity Propagation algorithm also does not require knowledge about the number of clusters. But unlike DBSCAN, which is a *density-based* clustering algorithm, affinity propagation is based on the idea of *passing messages between data points*. Calculating pairwise similarity based on some distance function (i.e. *Euclidean distance*) this algorithm then converges in some number of *standard representatives*. A dataset is then described using this small number of *standard representatives*, which are identified as the most representative instances on the particular cluster.



Affinity Propagation Clustering by Scikit Learn. Image source

The results of this algorithm often leave much to be desired, but it has a number of strong advantages. However, its main disadvantage, the computational complexity (caused by the need to calculate the distance for *all possible pairs of data points*) does not allow it to be used on large datasets.

Pros:

- Knowledge about the number of clusters is not necessary;
- As a result, we also have *standard representatives* of a cluster. Unlike K-Means centroids, these instances are not just average values, but real objects from the dataset.

Cons:

- Works much slower than other algorithms due to computational complexity;
- Does not cope well with a large number of instances;
- Separates only convex and homogeneous clusters well.

Mean Shift

The Mean Shift algorithm first places a circle of a certain size (radius of the circle is a parameter called `bandwidth`) in the center of each data point. After that, it iteratively calculates the *mean* for each circle (the average coordinates among the points inside the circle) and *shifts* it. These *mean-shift* steps are performed until the algorithm converges and the circles stop moving.

You can see Mean Shift algorithm visualization here.

Mean Shift converges into local regions with maximum density. Then all the circles that are close enough to each other form a cluster. Therefore, at the same time, it solves the *density estimation* task and calculates cluster centroids.

As DBSCAN, this algorithm represents a **density-based approach** so works badly with sparse data.

Pros:

- Knowledge about the number of clusters is not necessary;
- Have just one hyperparameter: the radius of the circles;
- Solves *density estimation* task and calculate cluster centroids;
- Does not find a cluster structure where it is not actually present.

Cons:

- Does not cope well with sparse data and with a large number of features;
- Does not cope well with a large number of instances;
- Does not cope well with clusters of complex shapes: tends to chop these clusters into pieces.

BIRCH

The BIRCH stands for *Balanced Iterative Reducing and Clustering using Hierarchies*.

This hierarchical clustering algorithm was designed specifically for large datasets. In the majority of cases, it has a computational complexity of *O(n)*, so requires only one scan of the dataset.

During training, it creates a dendrogram containing enough information to quickly assign each new data instance to some cluster without having to store information about all instances in memory. These principles allow getting the best quality for a given set of memory and time resources compared with other clustering algorithms. They also allow to incrementally cluster incoming data instances performing *online learning*.

Pros:

- Was designed specifically for very large datasets;
- Show the best quality for a given set of memory and time resources;
- Allows implementing online clustering.

Cons:

- Does not cope well with a large number of features.

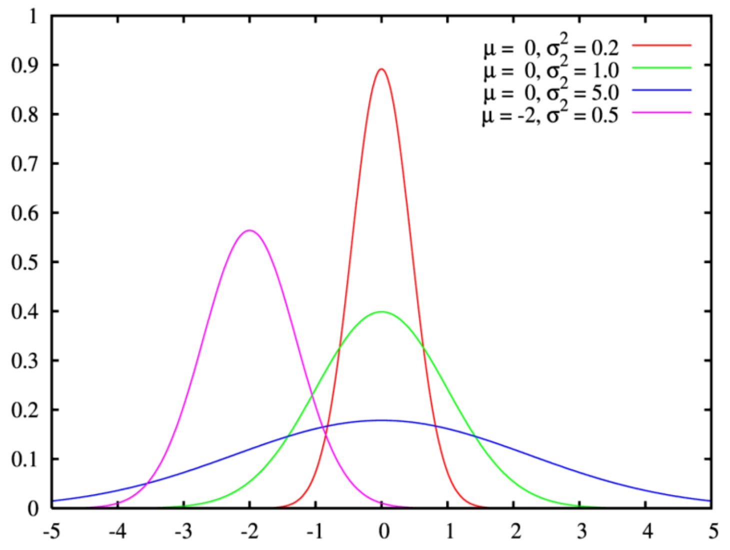
Gaussian Mixture Models

Gaussian Mixture Models (*GMM*) is a probabilistic algorithm that can solve as many as three unsupervised learning tasks: *clustering*, *density estimation*, and *anomaly detection*.

This method is based on the *Expectation Maximization* algorithm and assumes that data points were generated from a group (*mixture*) of Gaussian distributions. This algorithm can result in poor local solutions, so it needs to be run several times keeping only the best solution (`n_init` parameter in *sklearn*).

It is known that in the general case the Gaussian distribution has two parameters: a vector of the *mean* (μ) and a matrix of *variance* (σ^2). Then, if it is known that the data can be divided into `N` clusters in `M`-dimensional space, the task of the algorithm is to select `N` μ vectors (with `M` elements) and `N` σ^2 matrices (with `MxM` elements).

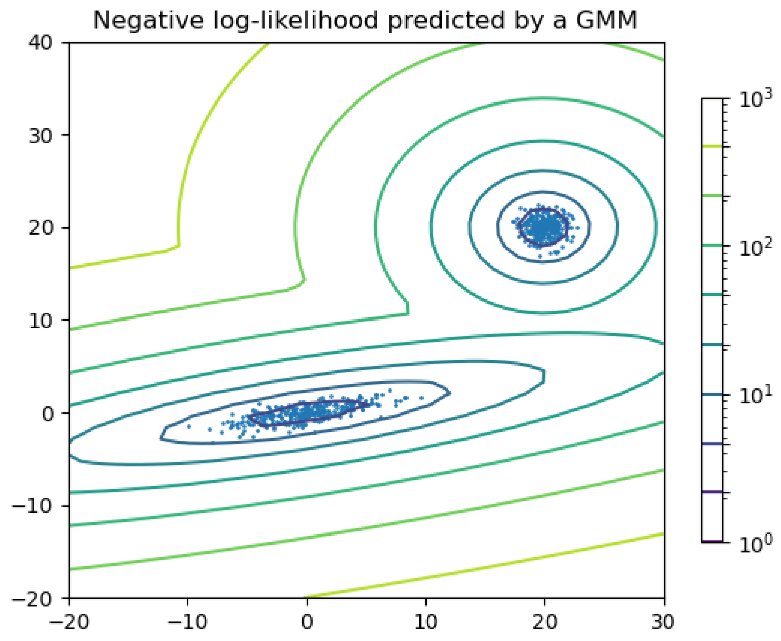
In the case of one-dimensional space, both μ and σ^2 are scalars (single numbers).



***Gaussian (normal) distribution with different parameters in one-dimensional space.* [*Public domain*]**

In the image below you can see two distributions in two-dimensional space. Each of the distributions has the following parameters:

- two values for the mean vectors (x and y coordinates);
- four values for the variance matrices (variances in main diagonal and covariances in the other elements).



***Density Estimation with GMM by Scikit Learn.* [*Image source*]**

To choose a good number of clusters you can use *BIC* or *AIC* (Bayesian / Akaike information criterion) as metrics and choose the model with its minimum value. On the other hand, you can use the **Bayesian GMM** algorithm. This model only requires a value that is greater than the possible number of clusters and can detect the optimal number itself.

Also, the Gaussian Mixture Model is a *generative model*, meaning that 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 datasets that were generated from a mixture of Gaussian distributions with different shapes and sizes;
- At the same time solves *density estimation* and *anomaly detection* tasks;
- Is a *generative model*, so can generate new instances.

Cons:

- Knowledge about the number of clusters is necessary and must be specified as a parameter (not in the case of *Bayesian GMM*);
- *Expectation Maximization* algorithm can result in poor local solutions, so it needs to be run several times;
- Does not scale well with large numbers of features;
- Assume that data instances were generated from a mixture of Gaussian distributions, so cope badly with data of other shapes.

How to choose a clustering algorithm?

As you can see, the 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 as a parameter (*DBSCAN* or *Affinity Propagation*) and use the resulting value as a starting point.

Another important question remains the evaluation of quality - you can try all the algorithms, but how to decide 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. Understanding these metrics and how to use them successfully comes with experience and goes beyond the scope of this article.

Other Unsupervised Learning Tasks

Although dimensionality reduction, anomaly detection, and clustering are the main and the most popular unsupervised learning tasks, there are others.

Since the definition is blurry, any algorithm that deals with an unlabeled dataset can be considered solving some unsupervised learning task (for example calculating the mean or applying Student's t-test). However, researchers often identify two other tasks among others: *Density Estimation* and *Association Rule Learning*.

Density Estimation

I have already briefly mentioned density estimation in the anomaly detection section.

Density Estimation is the task of estimating the density of the distribution of data points. More formally, it estimates the *probability density function* (PDF) of the random process that is generated by the given dataset. This task historically came from statistics, when it was necessary to estimate the PDF of some random variable and can be solved using statistical approaches.

In the modern era, it is used mostly for *data analysis* and as an auxiliary tool for *anomaly detection* - data points located in regions of low density are more likely to be anomalies or outliers. Now it is usually solved with *density-based clustering* algorithms such as **DBSCAN** or **Mean Shift**, and using *Expectation-Maximization* algorithm and **Gaussian Mixture Models**.

Association Rule Learning

Association Rule Learning (also called *Association Rules* or simply *Association*) is another unsupervised learning task. It is most often used in business analysis to maximize profits.

It aims to detect unobvious relationships between variables in a dataset, so also can be considered as a *data analysis* tool. There are many complex algorithms to solve it, but the most popular are:

- **Apriori** - based on breadth-first search;
- **Eclat** (*Equivalence Class Transformation*) - based on depth-first search; and
- **FP-Growth -** designed to detect frequently occurring patterns in the data.

A common example of such a task is *product placement*. For example, knowing that people often buy onions together with potatoes in supermarkets, it makes sense to place them side by side to increase sales. Therefore, associative rules are used in promotional pricing, marketing, continuous production, etc.

Conclusions

In this article, I tried to describe all the main unsupervised learning tasks and algorithms and give you a big picture of unsupervised learning.

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