Artificial Intelligence for Aerospace Engineers

Luca Magri,[1,2]

[1]Imperial College London, London, SW7 2AZ, UK
[2]Fellow of the Alan Turing Institute, London, NW1 2DB, UK

(LaTeX template inspired from annualreviews.org)

Keywords

Artificial intelligence, machine learning, optimization, data-driven methods, neural networks, aerospace engineering

Abstract

This is the first version of the notes I wrote in Sept 2021 for the course Artificial Intelligence for Aerospace Engineers at Imperial College London, Aeronautics Department. The course is open to a variety of students: 3^{rd}- and 4^{th}-year MEng students, as well as MSc students. The references cited in the notes are left for the keen readers, but they will not be examined. If you identify typos or mistakes (I am sure you will), please send me your corrections by e-mail: l.magri@imperial.ac.uk.

Contents

1. Lecture 7: Unsupervised learning

So far, we have learnt methods that need labelled datasets; in other words, we have learnt about supervised machine learning. The goal of today's lecture is to learn some methods that do not require labelled datasets; in other words, we will learn more about unsupervised learning.

1.1. Objectives

The objectives of this lecture are

1. understand the task of clustering;
2. learn the k-means algorithm, DBSCAN, and hierarchical clustering;
3. appreciate the task of anomaly detection;
4. learn the elliptic envelope method.

1.2. Unsupervised learning versus supervised learning

As we learnt in the first lecture, there are three paradigms for learning from data

- **Supervised learning**, in which the datasets are labelled, typically by a supervisor. Therefore, supervised learning markedly depends on the supervisor's intervention;

- **Unsupervised learning**, in which the training data is not labelled. The task of unsupervised learning is to find patterns within the data. Of course, the algorithm looks for patterns according to a prescribed principled (i.e., the machine needs to be set up by a supervisor), but, except for the *a-priori* prescription of the principle, the algorithm works on its own with little supervisor's intervention. Unsupervised learning can be used to find *patterns* in data, *cluster* and *classify* new data, as well as *reduce the dimensionality* of the problem by extracting the most important features. Unsupervised learning is the subject of this lecture.

- **Semi-supervised learning**, in which only a subset of the dataset is labelled.

Supervisor: A human being, also known as *teacher* or *expert*.

1.3. Classification versus Clustering

Classification and clustering are two tasks of machine learning.

On the one hand, *classification* sets *predefined* classes in which to put the features in. (You can think of these classes as boxes.) Therefore, in classification, there are predefined labels assigned to each input instance according to their properties. Classification is a supervised learning task.

On the other hand, *clustering* groups the instances based on their similarity without class labels. In contrast to classification, clustering *finds* the "boxes" into which classify data. Clustering is an unsupervised learning task.

Similarity: It is quantitatively measured by a metric (i.e., a distance function, such as a norm) in the feature space.

Common applications of clustering are

- **Customer segmentation**, in which customers are grouped (clustered) based on

their purchases, online presence, activity on social media, etc. This is useful to understand customers to design marketing campaigns to target each market segment (cluster);

- **Data analysis** to reduce the complexity of a datasets. This is useful in engineering, for example, for feature engineering and feature extraction;

- **Dimensionality reduction**, in which, once a dataset has been clustered, we measure each instance's *affinity* with each cluster. Each instance's feature vector can then be replaced with a vector of its cluster affinities. If there are k clusters, then this vector is k-dimensional. This vector has typically less components than the original feature vector, whilst preserving enough information for further analysis. Dimensionality reduction is useful in engineering, for example, in reduced order modelling of large-scale problems;

- **Anomaly detection**, also known as outlier detection, in which any instance that has a low affinity to all the clusters is likely to be an anomaly. Anomaly detection is particularly useful in detecting defects in manufacturing and malfunctions in structures (structural health monitoring);

- **Image segmentation**, in which pixels are clustered according to their colour. Hence, each pixel's colour is replaced with the mean colour of its cluster, which allows the image to be compressed because the number of different colours is reduced. Image segmentation is used in object detection and tracking systems, in which it is necessary to detect the contours of the objects to track.

Feature engineering: A machine learning task that finds patterns in data to create new variables (features) that are not in the dataset. The new features can be used to create reduced-order models, for example.

Affinity: A measure of how well an instance fits into a cluster.

Reduced-order modelling: Technique to reduce the computational complexity of a problem. For example, you can reduce the complexity of Navier-Stokes equations by projecting them onto the most important principal components.

What is a cluster?

There is no universal definition of cluster. The definition depends on the context. Different algorithms capture different types of clusters. For example, some algorithms seek for instances centered around a particular point, called a centroid (e.g., k-means algorithm). Other algorithms seek continuous regions of densely packed instances (e.g., DBSCAN). Other algorithms are hierarchical, which seek clusters of clusters (e.g., agglomerative and divisive algorithms).

Example on clustering versus classification

The Iris flower dataset (https://en.wikipedia.org/wiki/Iris_flower_data_set.) includes measurements of 150 irises of three varieties: setosa, versicolor, and virginica. Each flower has 50 samples with measurements of the plant biology (sepal length, sepal width, petal length, and petal width). Figure 1 shows three features, which are sufficient for clustering and classification because the three iris varieties are separated. The versicolor and virginica have a small overlap among the samples taken. Although for this data set machine learning is not required (we can distinguish the classes by inspection), engineering systems' data rarely reduces to a small number of features. In classification, we would feed the trained machine learning algorithm, say a neural network, with new instances. The machine would predict into which of the three classes the new instance belongs. In contrast, in clustering, we do not label the data. The machine finds the clusters by learning similarities between the instances according to different criteria, as explained in today's lecture.

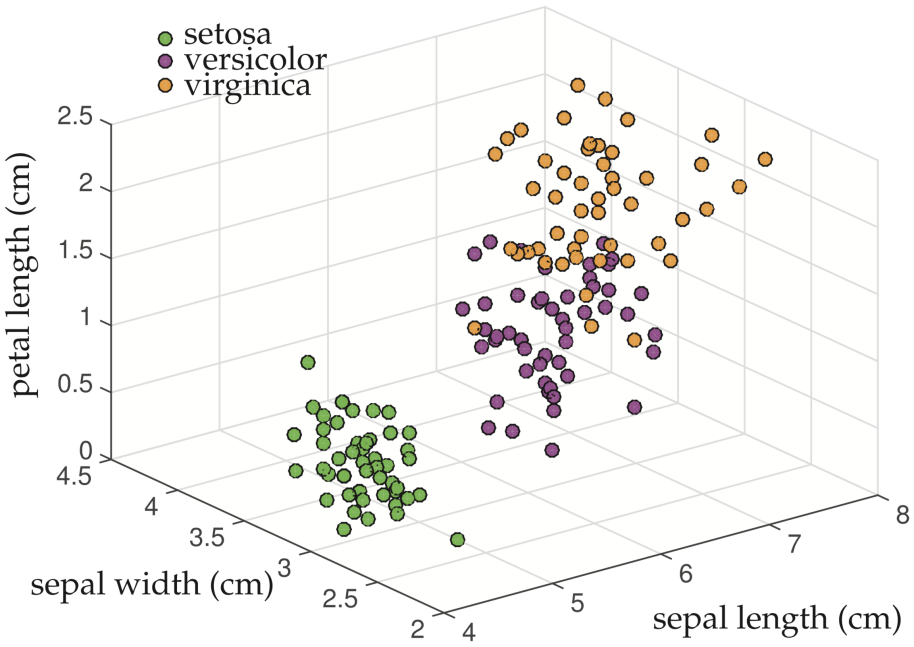

Figure 1: Iris flower dataset. Each flower includes a measurement of sepal length, sepal width, petal length, and petal width. Source: Brunton & Kutz (2019).

1.4. k-Means clustering

Heuristic: A heuristic algorithm finds a solution of a complicated problem in a reasonable time frame. The solution may not be an optimal solution, but it is a good trade-off between optimality and execution time.

The k-means clustering algorithm, also knows as Lloyd algorithm or LLoyd-Forgy algorithm, is one of the most common clustering techniques. It is a heuristic algorithm. We are given a set of vector-valued data, $\mathbf{x}^{(i)}$, with the goal of grouping m observations into k clusters. Each observation is placed in the cluster with the nearest mean. This results in a partitioning of the data space into Voronoi cells. Although the number of observations and dimensions of the system are known, the number of partitions k is generally unknown and must also be found. The k-means algorithm is iterative. Loosely put, the algorithm proceeds as follows: (i) given initial values for k distinct means, compute the distance of each observation, $\mathbf{x}^{(i)}$, to each of the k-means; (ii) label each observation as belonging to the nearest mean; and (iii) find the baricentre (mean) of each group of labelled points. These new means are then used to iterate back from step (i) in the algorithm (Figures 2,6,8).

In detail (Figure 2),

1. **Choose a metric (distance function)**. This quantitatively defines the notion of *similarity* by measuring the distance between points *in the feature space*: The closer (with respect to the metric) the points are, the more similar. (Remember: The points live in the feature space, whose coordinates can be completely unrelated to spatial variables.) We choose the squared Euclidean distance in this lecture, $||\cdot||^2$.

2. **Choose number of clusters,** k.

3. **Choose the centroids.** These are the baricentres (means), $\boldsymbol{\mu}_j$ of the clusters, which represent the positions of the clusters.

4. **Compute the distances** of each instance with respect to each centroid, $||\mathbf{x}^{(i)} - \boldsymbol{\mu}_j||^2$ for $i = 1, \ldots, m$ and $j = 1, \ldots, k$.

5. **Assign** each instance, $\mathbf{x}^{(i)}$ to the closest centroid, j.

6. **Compute the baricentre** (mean) of each centroid, $\boldsymbol{\mu}_{j,\text{new}} = 1/N_j \sum_{j=1}^{N_j} \mathbf{x}^{(j)}$, where N_j is the number of points, $\mathbf{x}^{(j)}$, that belong in cluster j. Therefore, $\sum_{i=1}^{k} N_i = m$.

7. **Update centroid**. The new centroid is the baricentre from the previous instruction, $\boldsymbol{\mu}_{j,\text{new}}$.

8. **Compute the new distances** from the new centroids, $||\mathbf{x}^{(i)} - \boldsymbol{\mu}_{j,\text{new}}||^2$ for $i = 1, \ldots, m$ and $j = 1, \ldots, k$.

9. **Does the mean distance from the centroid change?** If so, repeat steps 5-8. If not, stop.
   (a) Another termination condition that might be used is "Is any point re-assigned to a different cluster"? If so, repeat steps 5-8. If not, stop.
   (b) Alternatively, "Is the user-defined number of maximum iterations reached?". If not, repeat steps 5-8. If so, stop.

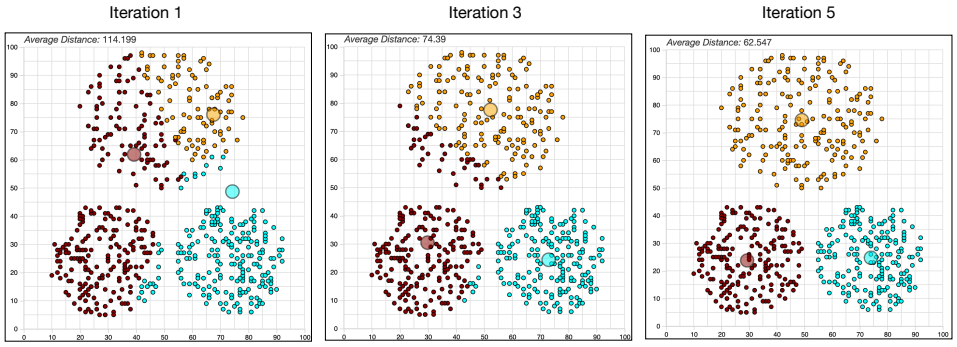

Figure 2: k-means clustering: The process.

1.4.1. k-means as an optimization problem. Formally, the objective is to maximize the distances between centroids whilst minimizing the distances of the instances from the centroids, which is the solution of an optimization problem

$$\text{Find centroids} \quad \boldsymbol{\mu}_j \quad \text{to minimize} \tag{1.}$$

$$\sum_{j=1}^{k} \sum_{i=1}^{N_j} ||\mathbf{x}^{(i)} - \boldsymbol{\mu}_j||^2, \tag{2.}$$

where, as mentioned above, N_j is the number of points, $\mathbf{x}^{(i)}$ in cluster, j. The solution of 2 minimizes the sum of the variances within the clusters. The k-means algorithm is guaranteed to converge to a solution because the mean-squared distances between instances and centroids can only decrease at each iteration.

Computational complexity of k-means

Finding the general optimal solution of 2 is nearly an intractable problem. This *practically* means that an algorithm to solve it has a running time that can be exceedingly long (NP-hard to be precise). The k-means algorithm will converge to a solution, although it is not guaranteed that the solution will be the globally optimal solution. If the data has a clustered structure, the computational complexity of the k-means algorithm is linear with the number of instances, m, the number of clusters, k, and the number of dimensions, n (i.e., the number of the components of $\mathbf{x}$). If the data does not have a clustered structure, the computational complexity can (theoretically) become exponential. In practice, the k-means algorithm converges quickly to a local minimum for most problems.

1.4.2. Limits. k-means clustering is fast, effective, and scalable. It is one of the top 10 algorithms in Data Mining 2008. However, k-means has some limits: (i) although the algorithm is guaranteed to converge, it may not converge to the globally optimal solution because of the *centroid initialization* and the *a-priori* choice on the *number of clusters*; and (ii) it not does not perform well when the clusters have varying sizes, different densities, or nonspherical shapes.

1.4.3. Performance metric. A good performance metric to judge the quality of a k-means solution with respect to another random initialization for a given number of clusters is the average mean-squared distance between each instance and the corresponding centroid, which is also know as *inertia*. Physically, you can think of a cluster as being composed as points with the same mass: the larger the distance of the masses, the larger the (moment of) inertia (Figure 4).

1.4.4. Strategies. We have some strategies to improve the solutions:

- **Centroid initialization**. A simple, yet efficacious, solution is to run the algorithm with different random initializations for the centroid locations (Figure 3). How many random initializations? This is a trade-off between how much time you have and by how much you want the solution to improve. Ten could be a good number if you have no clue as to how to start.

- **Number of clusters.** Whereas the inertia is a good metric for judging the improvement of a centroid initialization, it is not a good metric for judging the improvement of increasing the number of clusters. This is because the inertia consistently decreases as the number of clusters increases: The more centroids (i.e., clusters) you have, the closer each instance is to the centroid, hence, the smaller the inertia. A simple, yet good, method is to plot the inertia as a function of the number of clusters, and choose the number of clusters at the inflection point (elbow). This is called the *elbow method* (Figure 4).

- **Feature scaling**. You should scale the input features before you run k-means, or the clusters may be stretched, with k-means performing slowly and poorly. Scaling the features does not guarantee that all the clusters will be spherical, but it improves the clusters' shapes for k-means to perform more effectively. Alternatively, use DBSCAN, which is Density-Based Clustering of Applications with Noise (Section 1.5).

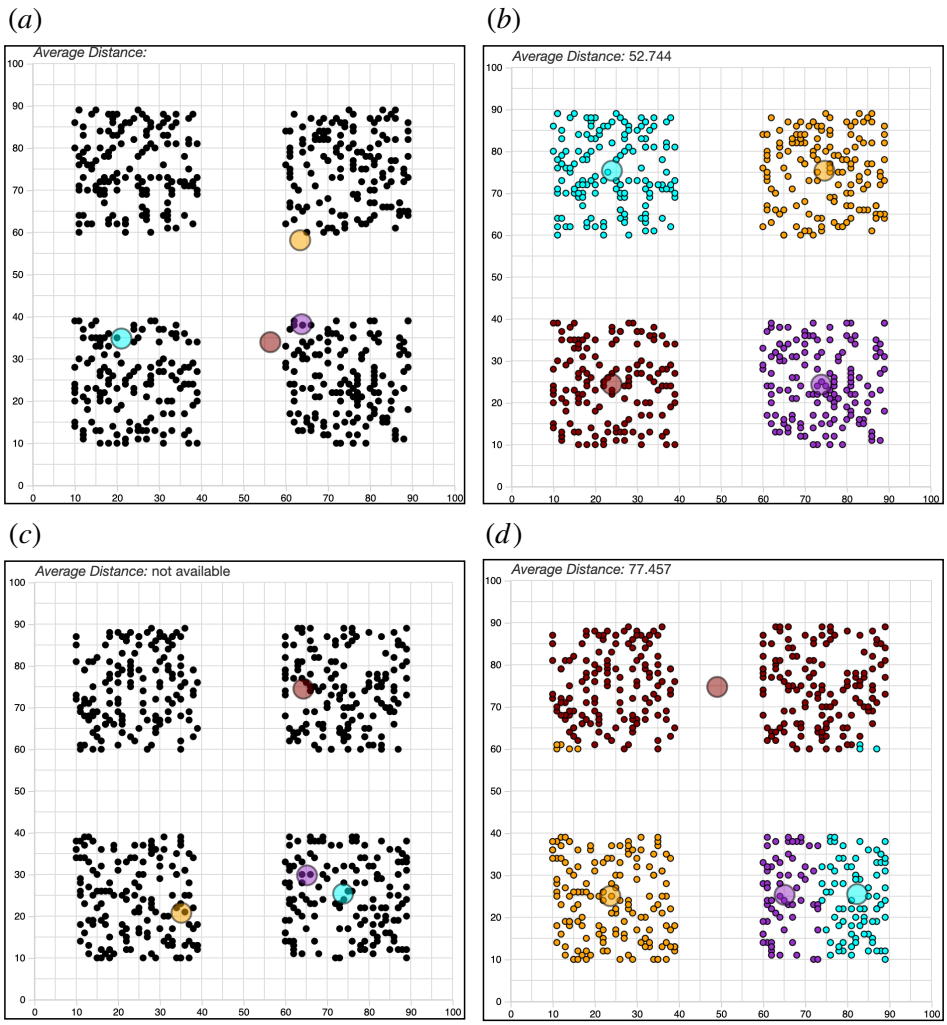

Figure 3: k-means clustering finds a solution heuristically. The solution may depend on the centroid initialization. (a) Initial guess on the centroids, and (b) final converged solution. (c) Another initial guess on the centroids. (d) Final converged solution, which is less optimal than (b).

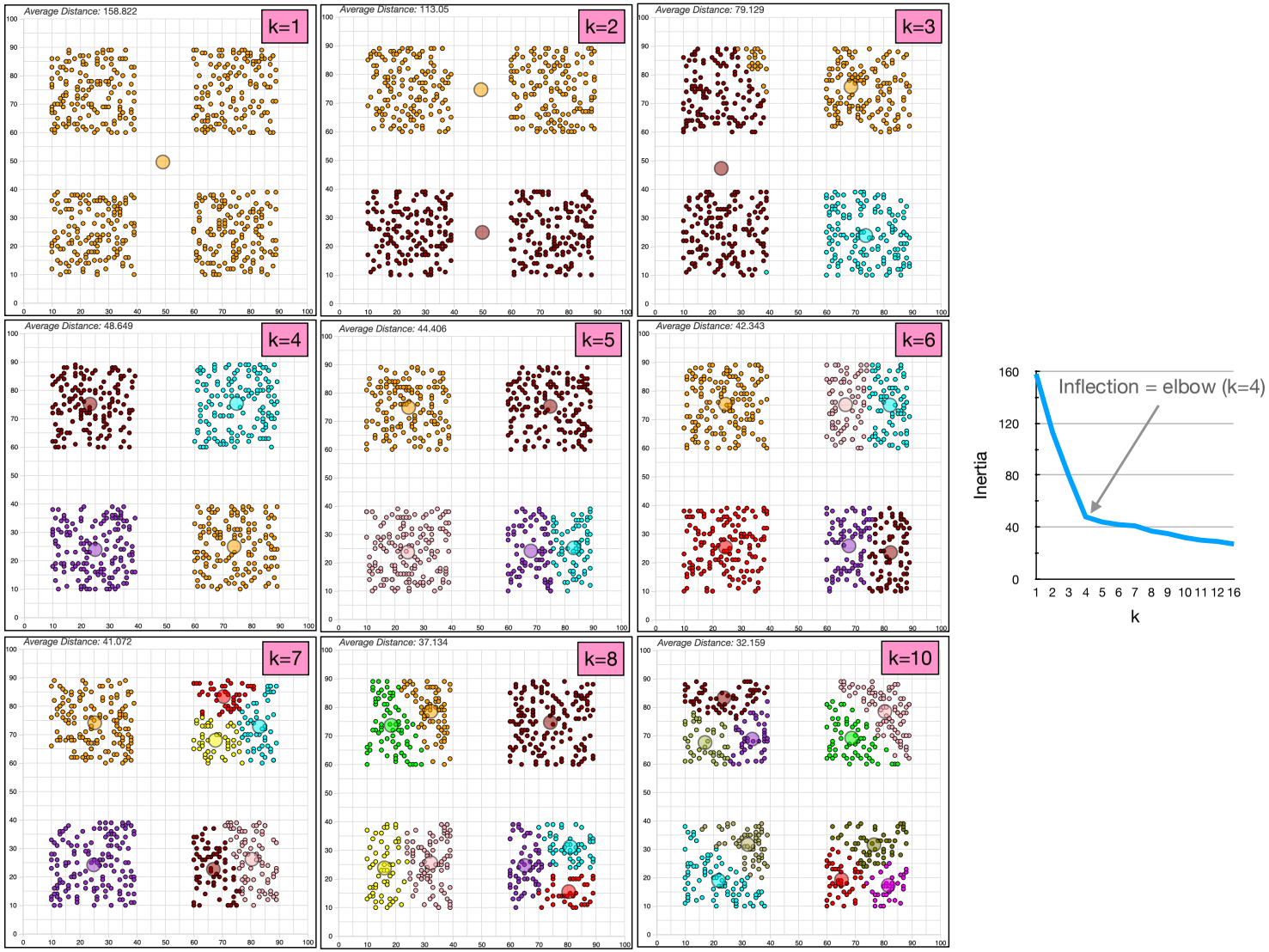

Figure 4: The elbow method is a simple method to select the optimal number of clusters. This number should be chosen at the "elbow" of the right panel, which is the point at which the derivative significantly changes in magnitude.

1.5. Density-based spatial clustering of applications with noise (DBSCAN)

Density-based spatial clustering of applications with noise (DBSCAN) is a data clustering algorithm, which uses a different logic than k-means. In fact, DBSCAN is density-based, rather than centroid-based. Given a set of points in some feature space, DBSCAN groups points that are closely-packed together (points with many nearby neighbours). It identifies points that are isolated in low-density regions as outliers. (See Figures 6,8 for hands-on examples.) Loosely put, DBSCAN defines clusters as continuous regions of high density (Figure 5). In detail,

1. **Choose a metric (distance function)**. We choose the squared Euclidean distance in this lecture, $||\cdot||^2$.

2. **Choose** ε to define the concept of vicinity to other points, i.e., the neighbourhood $||\mathbf{x}^{(i)} - \mathbf{x}^{(j)}||^2 \leq \varepsilon$. Choose N to define the number of points that are needed to classify a neighbourhood as high density.

3. **Select** a new instance.

4. **Count** how many instances are located within a small distance ε from it (the instance's ε-neighbourhood).

   (a) If an instance has at least N instances in its ε-neighbourhood (including itself), then it is considered a *core instance*, which means that it is a dense region. All instances in the ε-neighbourhood of a core instance belong to the same cluster. This neighbourhood may include other core instances; therefore, a long sequence of neighbouring core instances forms a single cluster.

   (b) If fewer than N points belong to a neighbourhood, the region is not a *core* region (it has low density). Therefore, return to step 3.

5. **Classify** as anomalies (i.e., noise) all instances that are not core instances and do not have a core instance in their neighbourhoods.

The algorithm works well for problems that are characterized by densely-populated regions that are separated by lower-density regions.

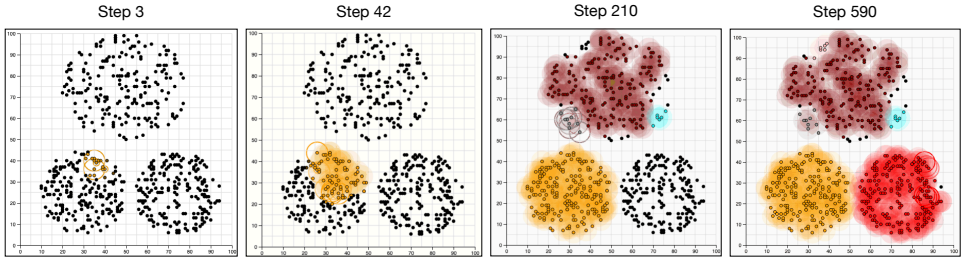

Figure 5: DBSCAN: The process.

1.5.1. Pros and cons. **Pros**: DBSCAN (i) does not need to specify the number of clusters, as opposed to k-means; (ii) can find clusters with complicated shapes, such as oblate spheroids; (iii) can classify instances as noise; and (iv) requires only two hyperparameters to tune, ε and N. **Cons**: DBSCAN (i) can misclassify border points that are reachable from multiple core clusters (the order with which the data is processed established in which cluster the border points belong); (ii) depends on the distance measure used to define the neighbourhood, like the k-means algorithm. The most common distance metric used is Euclidean distance; and (iii) does not cluster effectively when there are large differences in densities; and (iv) if the estimates on the data scales and distances are not good, it may be challenging to tune ε.

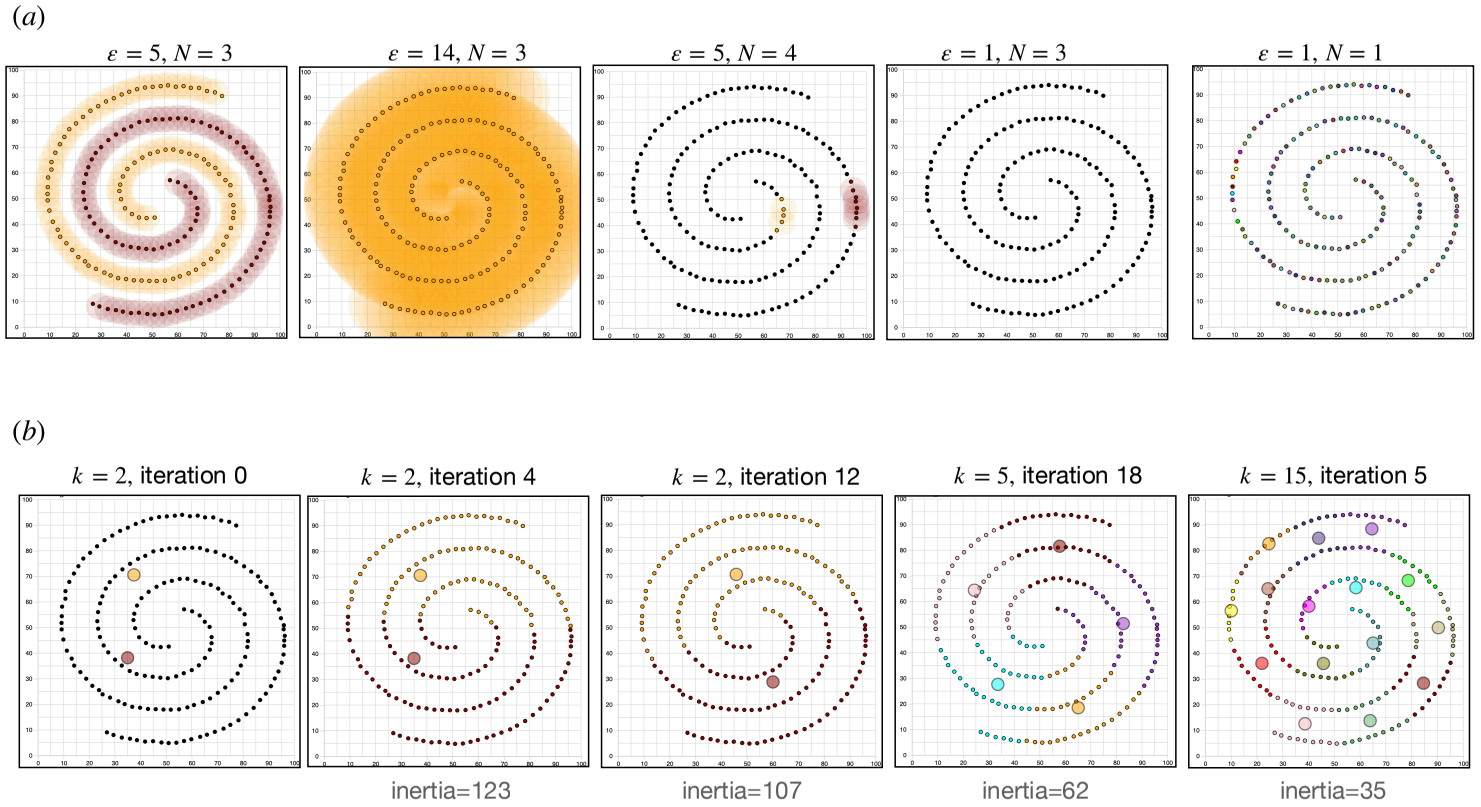

Figure 6: Example of (a) DBSCAN (b) and k-means in action on a spiral dataset for different values of the relevant hyperparameters. For hierarchical clustering, refer to the textbox on 15.

1.6. Hierarchical clustering

Greedy algorithm: It is based on a *divide and conquer* strategy. It breaks down a problem into simpler stages (subproblems), for which it finds the optimal result using heuristic. The solution of each stage (subproblem) becomes the input for the next stage.

$\mathcal{O}(m)$: The big-$\mathcal{O}$ notation describes the computational computational complexity of an algorithm, i.e., the approximate number of operations necessary to perform the instructions of an algorithm as a function of the number of inputs m.

Hierarchical clustering is a clustering algorithm, which builds a hierarchy of clusters. The hierarchical algorithm is *heuristic* and *greedy*. The results are visualized in *dendograms*. There are two broad strategies to build hierarchies (see Figures 7,8 for hands-on examples):

- **Bottom-up**, also known "agglomerative", in which each instance defines a cluster at iteration 0. The algorithm pairs and merges clusters, which become larger at every iteration. Here, we analyse the bottom-up approach, which is the most common.
- **Top-down**, also known as "divisive", in which all instances define a large cluster at iteration 0. The algorithm splits recursively the clusters down to smallest clusters.

The bottom-up hierarchical clustering consists of the following steps

1. **Choose a metric (distance function)**. We choose the squared Euclidean distance in this lecture, $||\cdot||^2$.
2. **Compute** the distances between all the instances, $||\mathbf{x}^{(j)} - \mathbf{x}^{(i)}||^2$ for $i, j = 1, \ldots, m$.
3. **Merge** the closest pair into a new cluster, which is located midway its original instances.
4. **Repeat** steps 1-2 with the remaining $m - 1$ instances.
5. **Stop** when you have only one large cluster that contains all the instances.

Example on bottom-up hierarchical clustering

In reference to Figure 7, the dataset consists of two spirals. Panels (a) show different hierarchies of the dendogram of panel (b). The horizontal red line in the dendogram is the highest hierarchy (panel (a)). There are 177 points in this dataset, therefore, the bottom part of the dendogram consists of 177 datapoints. First, the algorithm computes the mutual distances between all pairs of points, which requires $\sim \mathcal{O}(m^2)$ operations. In this case, it exactly requires the evaluation of $\sim 177^2 = 31329$ distances. Second, the closest pair is merged in a new cluster, with location that is midway of the two original locations. Now, we have $m - 1$ clusters. Third, we recalculate the mutual distances between the pairs of points, which requires $\sim 176^2 = 30976$ operations. We select the closest pair, and merge the two points in one new cluster as before. We iterate until we have only one large cluster left (panel (a)).

What is the optimal number of clusters? In this case, we know already by inspection (the answer is two) because the dataset is simple. But, how can we automatically find the number of clusters? There is no definitive answer because cluster analysis is a heuristic approach. The interpretation of the hierarchical structure depends on the context. Often, a number of solutions are good from a theoretical point of view. However, there are methods that help make a decision on the number of clusters, for example, silhouette plots (p. 247, Géron 2019), elbow methods, and heights of the vertical lines of the dendograms (which represent the distances between the connected clusters). Silhouette plots are not explained because they are not in the exam.

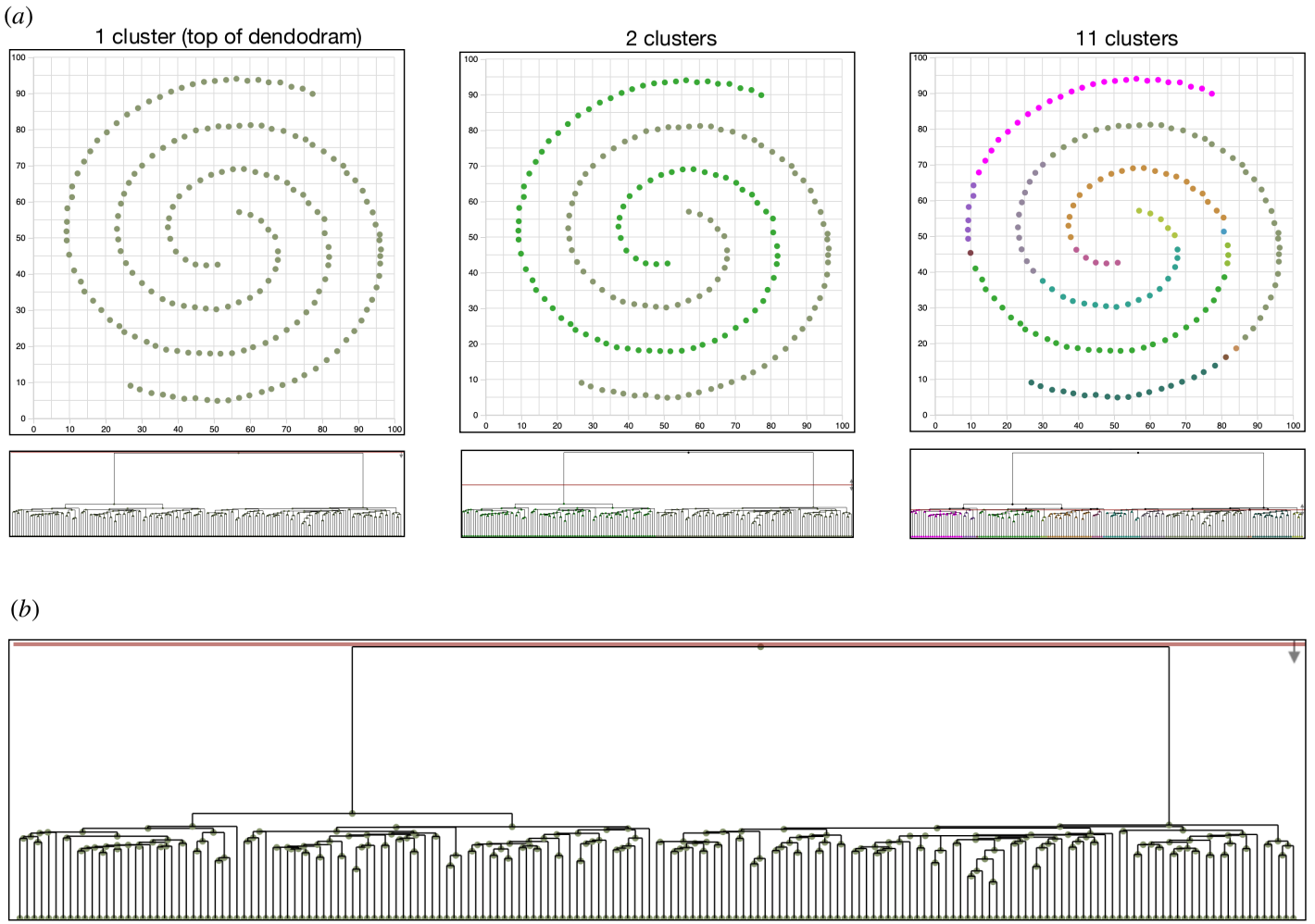

Figure 7: Example of hierarchical clustering (bottom-up). (a) represent clusters indicated by the horizontal red lines in the dendograms. (b) is the dendogram, which is a zoomed-in version of the small dendograms below the panels (a). The number of clusters is the number of vertical lines that are intersected by the red line defined as the threshold. Please, refer to the textbox on 15.

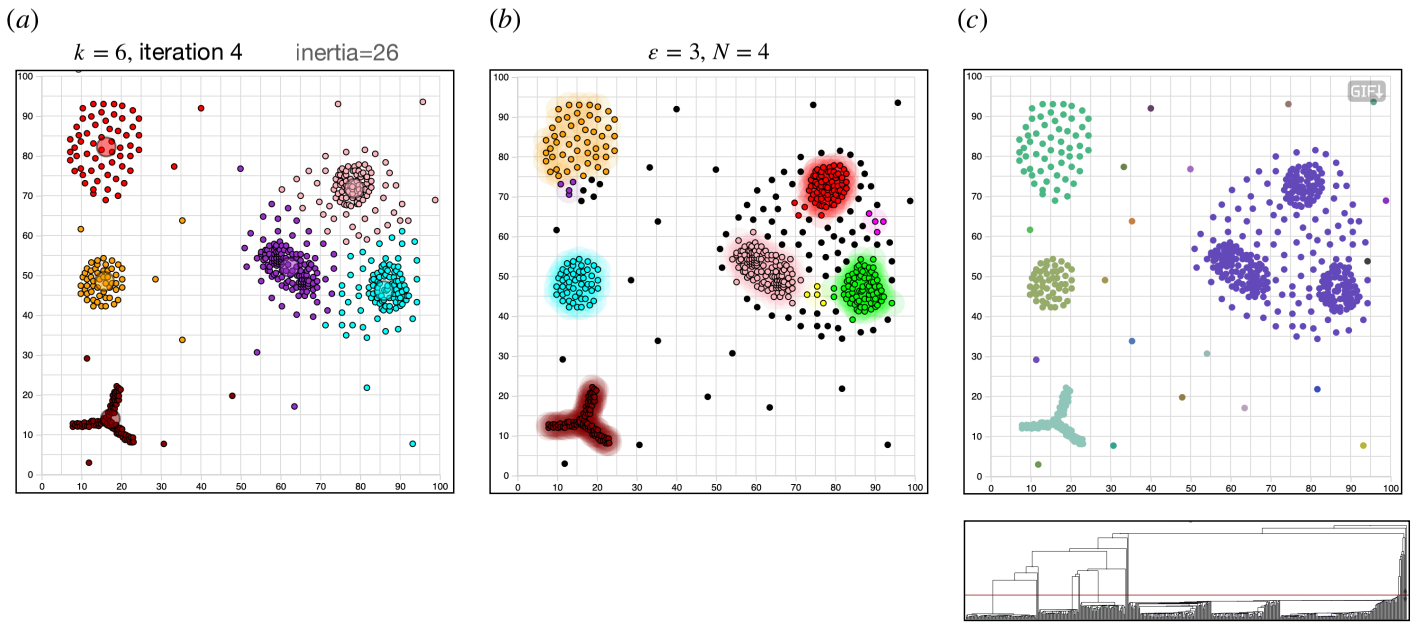

Figure 8: Example of a dataset with different shapes and densities. (a) k-means with six clusters; (b) DBSCAN with $\varepsilon = 3$ and $N = 4$; (c) hierarchical clustering with the dendogram. The hyperparameters k, ε, and N have been tuned before making this figure. Hierarchical clustering is non-parametric, i.e., it requires only the definition of a distance function, but it requires no hyperparameter.

1.7. Elliptic envelope for anomaly detection

The Gaussian density, named after the German mathematician Carl Friedrich Gauss (1777-1855), is of central importance in the study of probability and machine learning. It is a good model (sometimes provably accurate) for many quantities, such as velocities of particles in an ideal gas, noise added to a signal in communications, measurement noise, height and weights in human beings, price indices, etc.

Aside: Gaussian or normal?

The Gaussian density is also called the normal density. The name was introduced by the English mathematician Karl Pearson who believed that other mathematicians than Gauss deserved to share the credit for its invention. He later regretted the name, because it may lead people to believe that other densities are abnormal. Gauss's first use of the density is undisputed nowadays and all the mathematicians involved in Pearson's attempted historical re-engineering have had important mathematical concepts named after them (e.g., the Laplace transform). Sadly, the name "normal" has stuck, and it is important for you to be aware of this alternative name as you may encounter it frequently.

1.7.1. Monovariate Gaussian distribution. A random variable, X, follows a Gaussian distribution if its probability density function (PDF) is

$$f_X(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right), \tag{3.}$$

$$\sim \mathcal{N}(\mu, \sigma^2). \tag{4.}$$

where μ is the mean (expectation, $E(X)$), σ^2 is the variance ($E(X^2) - E(X)^2$), and $\mathcal{N}$ is a shorthand for "Normal". Therefore, $f_X(x) \sim \mathcal{N}(\mu, \sigma^2)$ means that the random variable X follows a Gaussian distribution with mean μ and variance σ^2. The mean and variance are the only two parameters that define a Gaussian distribution for one variable (Figure 9).

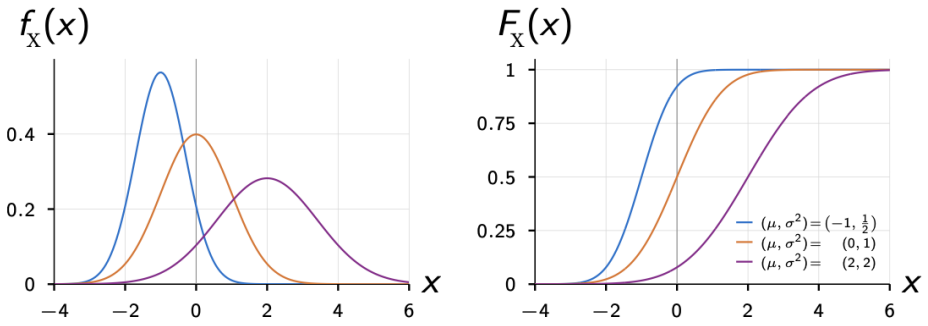

Figure 9: Gaussian probability density function (left) and cumulative density function (right) for different means and variances. The cumulative function is the integral of the probability density function.

1.7.2. Multivariate Gaussian distribution. A column-vector of random variables (features), $\mathbf{X} = [X_1, X_2, \ldots, X_N]^T$ is Gaussian distributed if it follows the Gaussian probability density function

$$f_{\mathbf{X}}(\mathbf{x}) = \frac{1}{\sqrt{(2\pi)^N \det(\boldsymbol{\Sigma})}} \exp\left(-\frac{||\mathbf{x}-\boldsymbol{\mu}||^2_{\boldsymbol{\Sigma}}}{2}\right), \tag{5.}$$

$$\sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}), \tag{6.}$$

where $\boldsymbol{\mu}$ is the column-vector of the means, which can be computed from a sample, m, as

$$\boldsymbol{\mu} = \frac{1}{m}\sum_{i=1}^{m} \mathbf{x}^{(i)} \qquad \text{or, equivalently}$$

$$\mu_j = \frac{1}{m}\sum_{i=1}^{m} x_j^{(i)}, \qquad j = 1, 2, \ldots, N, \tag{7.}$$

$\boldsymbol{\Sigma}$ is the covariance matrix (Figure 10), which can be computed from a sample, m, as

$$\boldsymbol{\Sigma} = \frac{1}{m-1}\sum_{i=1}^{m}(\mathbf{x}^{(i)} - \boldsymbol{\mu})(\mathbf{x}^{(i)} - \boldsymbol{\mu})^T, \text{or, equivalently}$$

$$\Sigma_{ij} = \begin{bmatrix} \sigma_1^2 & \frac{\sum(x_1^{(i)}-\mu_1)(x_2^{(i)}-\mu_2)}{m-1} & \cdots & \cdots & \frac{\sum(x_1^{(i)}-\mu_1)(x_N^{(i)}-\mu_N)}{m-1} \\ \frac{\sum(x_1^{(i)}-\mu_1)(x_2^{(i)}-\mu_2)}{m-1} & \sigma_2^2 & & & \frac{\sum(x_2^{(i)}-\mu_2)(x_N^{(i)}-\mu_N)}{m-1} \\ \vdots & \vdots & & \ddots & \vdots \\ \frac{\sum(x_1^{(i)}-\mu_1)(x_N^{(i)}-\mu_N)}{m-1} & \cdots & & \cdots & \sigma_N^2 \end{bmatrix} \tag{8.}$$

Unbiased sample covariance: The sample covariance matrix has $m-1$ in the denominator, and not m. It can be proved than this provides an unbiased estimator of the covariance (Bessel's correction).

(Note that the Σ inside the matrix is a sum over $i = 1, 2, \ldots, m$, which should not be confused with the covariance matrix Σ_{ij}.) The covariance matrix is a generalization of the variance to multivariate problems. It is a positive semidefinite matrix (i.e., with non-negative eigenvalues), and it is diagonal only if the variables are statistically independent. Finally, the norm that measures the distance between the mean and a feature

$$||\mathbf{x}-\boldsymbol{\mu}||^2_{\boldsymbol{\Sigma}} \overset{\text{def}}{=} (\mathbf{x}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x}-\boldsymbol{\mu}), \tag{9.}$$

is also known as the *Mahalanobis distance*. The inverse of the covariance matrix, $\boldsymbol{\Sigma}^{-1}$, is also known as the *precision* matrix.

1.8. Elliptic envelope algorithm

The question we want to answer is "Given a test point, $\mathbf{x}^{(i)}$, is it an outlier (i.e., an anomaly)? Or is it what we should expect?" First, we need to make working assumptions on the distribution of the datapoints. In the elliptic envelope, we assume they are Gaussian distributed. Second, we compute the centroid (i.e., the mean) of the datapoints. Third, we compute the distance of the test point with respect to the mean of the distribution. Intuitively, the closer the test point, the more likely it belongs to the set, hence, the more

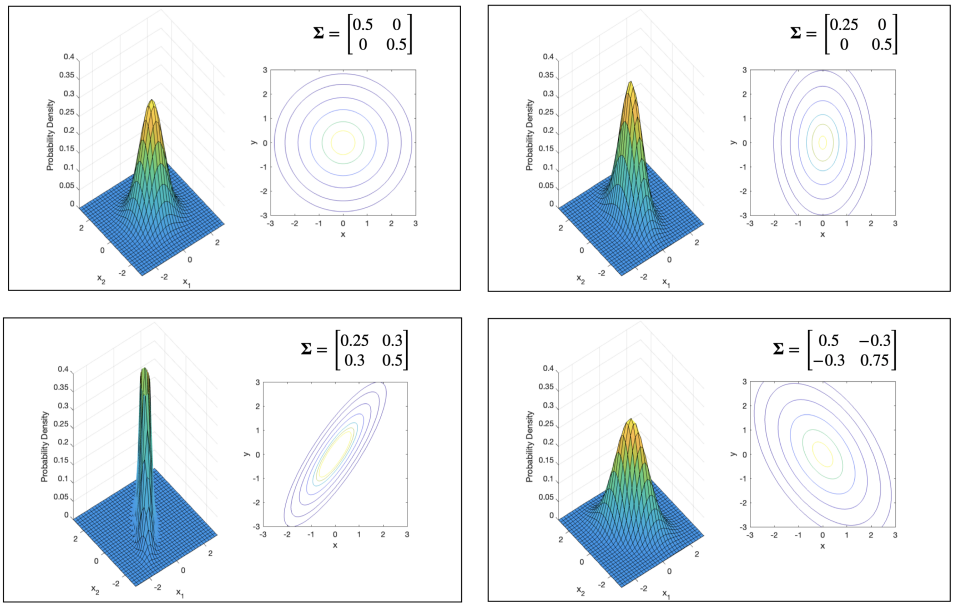

Figure 10: Bivariate Gaussian distributions with different covariances.

likely it is not an outlier. So, we need to measure the distance between a test point and a Gaussian distribution. Therefore, we compute $||\mathbf{x}^{(i)} - \boldsymbol{\mu}||$. Fourth, and crucially, we need to weigh the distance with the precision matrix, $||\mathbf{x}^{(i)} - \boldsymbol{\mu}||_{\boldsymbol{\Sigma}}$. The Mahalanobis distance is the distance of the test point from the center of mass divided by the width of the ellipsoid in the direction of the test point. Therefore, if the distribution is non-spherical (e.g., ellipsoidal), the probability of the test point depends both on the distance from the mean *and* the direction. In the direction of the short axis, the test point must be closer to the mean to be classified as a non-outlier. On the other had, in the direction of the long axis, the test point need to move farther away from the mean to be classified as an outlier. Fifth, we are ready to substitute the weighted-distance in the Gaussian distribution to evaluate the probability that test point belongs in the dataset.

To recap, the elliptic envelope algorithm is a distance-based anomaly detection method that can be summarized as

1. **Assume** the datapoints to be Gaussian distributed;
2. **Compute** mean and covariance;
3. **Compute** the Mahalanobis distance, D, between a test point and the mean;
4. **Choose** the decision boundary between "outliers" and "non-outliers", c;
5. **If** $D > c$, the test point is an outlier (i.e, anomaly), **otherwise** it is not. The decision boundary can be set $c = 1$ (which contains *roughly* the majority of the Gaussian-distributed datapoints), or other values.

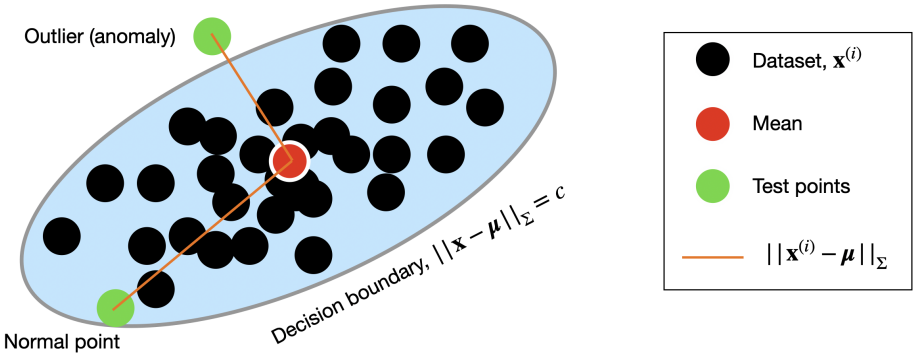

Figure 11: Elliptic envelope method to detect outliers (i.e., anomalies).

1.9. Exercises

1. For clustering, play with https://educlust.dbvis.de/# and https://scikit-learn.org/stable/modules/generated/sklearn.cluster.DBSCAN.html.
2. For Gaussians, play with https://distill.pub/2019/visual-exploration-gaussian-processes/.

1.10. Assignments 6

This needs to be submitted on Vocareum. The assignment The deadline for submission is on Tuesday 22nd March at 2.00 pm. No extensions will be allowed.

LITERATURE CITED

Brunton SL, Kutz JN. 2019. Data-driven science and engineering: Machine learning, dynamical systems, and control. Cambridge University Press

Géron A. 2019. Hands-on machine learning with scikit-learn, keras, and tensorflow: Concepts, tools, and techniques to build intelligent systems. O'Reilly Media