TASK Quarterly 28 (4) 2024

Generative methods in classification tasks

Rafał Lipiński *

Faculty of Electronics, Telecommunications and Informatics, Gdańsk University of Technology, Narutowicza 11/12, Gdańsk, 80-233, Poland
*rafal.lipinski@pg.edu.pl

https://doi.org/10.34808/tq2024/28.4/a

Abstract

The paper presents the implementation of generative methods in classification tasks. A distinction is made between two types of tasks—supervised learning and unsupervised learning—along with example use cases. Within the scope of supervised methods, the Bayes classifier and the use of the multivariate Gaussian distribution are described. To solve the unsupervised learning task using a generative approach, the Gaussian Mixture Model (GMM) is presented. The paper also describes a generative neural network based on an autoencoder architecture, implemented as a Variational Autoencoder (VAE).

Keywords:

Variational Autoencoder, Gaussian Mixture Model, Naive Bayes Classifier, Generative Classification

1. Introduction

Machine learning is a field of computer science in which algorithms are constructed based on input data. A typical task in this domain involves enabling a program to correctly assign input objects to predefined classes or to form abstract groups and associate each object with one of them. This problem is generally referred to as categorization.

Categorization methods are divided into supervised approaches, where the training set of objects is labeled with information about class membership. When the objects lack any information about their category, categorization is performed by automatically dividing them into groups, which is referred to as unsupervised learning.

The result of categorization is the assignment of objects to classes or categories, and this task can be addressed using a variety of approaches. In particular, algorithms can be distinguished as generative or discriminative [1].

Discriminative methods produce decision boundaries that define the separation between classes. In the simplest case, this boundary may be a straight line, with one class located above it and the other below. In many situations, such a classification method may provide sufficient results. Moreover, the resulting form is easy to interpret and leaves no doubt as the class to which a given object belongs. However, there is often a need for a more detailed description of the distribution of points in the space, which requires the use of more complex functions.

Generative methods build a model by describing the distribution of objects in the space through estimating the probability density function of class membership. In this case, we obtain properties of the groups not only at their boundaries but also within their interior.

2. Supervised Generative Methods

Supervised learning is characterized by the presence of information indicating whether the model's prediction is correct or not. This enables the algorithm to learn the relationships between features and classes, based on which it makes categorization decisions. This method requires data for which classes are labeled, which often involves costly human involvement.

2.1. Naive Bayes Classifier

The Bayes classifier [2] is a simple-to-implement probabilistic classifier based on Bayes' theorem, defined by Eq. 1.

$$p(A|B) = \frac{p(B|A)p(A)}{p(B)} \tag{1}$$

In the case of a feature vector $X = (x_1, x_2 ... x_n)$, for a given class C_k the above formula can be rewritten as shown in Eq. 2.

$$p(C_k|X) = \frac{p(X|C_k)p(C_k)}{p(X)} \tag{2}$$

The probability $p(C_k|X)$, known as the posterior probability, is calculated based on the observed data—specifically, the frequency of feature occurrences within objects of a given class. It represents the likelihood that an object characterized by features $x_1, x_2 ... x_n$ belongs to the class C_k.

The final classification is performed based on this value—the posterior probability is computed for each class, and the label of the class with the highest probability is selected.

Since the feature vector X is fixed and we are interested in whether the posterior probability of one class is greater than that of another, we can ignore the denominator and simplify Bayes' theorem to Eq. 3.

$$p(C_k|X) = p(X|C_k)p(C_k) \tag{3}$$

The variable $p(C_k)$ is called the prior probability and depends on the number of occurrences of objects in a given class relative to the total number of objects, as expressed by Eq. 4.

$$p(C_k) = \frac{\text{number of objects of class k}}{\text{total number of objects}} \tag{4}$$

The probability $p(X|C_k)$, using the chain rule, can be expanded into the product of the probabilities of features given the class, as shown in Eq. 6 [3].

$$\begin{aligned} p(X|C_k) &= p(x_1, x_2 ... x_n|C_k) \\ &= p(x_1|C_k)p(x_2, x_3 ... x_n|C_k) \\ &= p(x_1|C_k)p(x_2|C_k)p(x_3, x_4 ... x_n|C_k) \\ &= p(x_1|C_k)p(x_2|C_k)...p(x_n|C_k) = \prod_{i=1}^{n} p(x_i|C_k) \end{aligned} \tag{5}$$

Eq. 6 represents the product of the probabilities of occurrence of each feature $x_1, x_2 ... x_n$ within class C_n. Each of these probabilities can be calculated using the normal distribution, where μ_{ik} and σ_{ik} denote the mean and standard deviation of feature x_i in class C_k, respectively.

$$p(x_i|C_k) = \frac{1}{\sigma_{ik}\sqrt{2\pi}} exp(\frac{-(x - \mu_{ik})^2}{2\sigma_{ik}^2}) \tag{6}$$

Ultimately, we obtain Eq. 7, which defines the probability of belonging to class C_k based on the feature vector X.

$$\begin{aligned} p(C_k|X) &= p(C_k)p(x_1|C_k)p(x_2|C_k)...p(x_n|C_k) \\ &= p(C_k)\prod_{i=1}^{n}p(x_i|C_k) \end{aligned} \tag{7}$$

2.1.1 Example

To illustrate the operation of the Naive Bayes classifier, let us consider the set of objects presented in Tab. 1, described by four features and belonging to one of two classes.

Table 1: Training data

Class	feature x_1	feature x_2	feature x_3	feature x_4
0	2.3	1.2	4.5	0.81
0	3.3	1.8	3.8	0.32
1	1.2	2.3	3.1	0.31
1	1.8	2.1	2.6	0.13
1	2.1	2.8	3.2	0.02

For these data, we calculate the mean and standard deviation of each feature within each class. For example, the mean μ_{x_1} for class 0 represents the average value of feature x_1 in class 0: $\mu_{x_1} = \frac{2.3+3.3}{2} = 2.8$. σ_{x_1} for class 0 denotes the standard deviation of feature x_1 values in class 0: $\sigma_{x_1} = \sqrt{\frac{(2.3-2.8)^2+(3.3-2.8)^2}{2}} = 0.5$. The mean and standard deviation values for each feature in both classes are shown in Tab. 2.

Table 2: Calculated data

Class	μ_{x_1}	σ_{x_1}	μ_{x_2}	σ_{x_2}	μ_{x_3}	σ_{x_3}	μ_{x_4}	σ_{x_4}
0	2.8	0.5	1.5	0.18	4.15	0.245	0.565	0.120
1	1.7	0.21	2.4	0.13	2.96	0.103	0.153	0.0214

The prior probability of classes $p(C_k)$ can be calculated using Eq. 4.

$$p(C_0) = \frac{\text{number of objects of class 0}}{\text{total number of objects}} = \frac{2}{5} \tag{8}$$

$$p(C_1) = \frac{\text{number of objects of class 1}}{\text{total number of objects}} = \frac{3}{5} \tag{9}$$

Let us consider a feature vector $X = (x_1, x_2, x_3, x_4)$, which is to be classified into one of the classes.

Table 3: Object to be classified

Class	feature x_1	feature x_2	feature x_3	feature x_4
?	2.5	1.9	3.2	0.1

The probability of the vector belonging to class 0 can be calculated using Eq. 7. The prior probability is already known, and the missing values $p(x_1|C_0)$ and $p(x_2|C_0)$ can be computed using the normal distribution, as shown in Eq. 6.

$$\begin{aligned} p(x_1|C_0) &= \frac{1}{\sigma_{x_1C_0}\sqrt{2\pi}}exp(\frac{-(x-\mu_{x_1C_0})^2}{2\sigma^2_{x_1C_0}}) \\ &= \frac{1}{0.5\sqrt{2\pi}}exp(\frac{-(x-2.8)^2}{2\cdot 0.5^2}) = 0.665 \end{aligned} \tag{10}$$

$$\begin{aligned} p(x_2|C_0) &= \frac{1}{\sigma_{x_2C_0}\sqrt{2\pi}}exp(\frac{-(x-\mu_{x_2C_0})^2}{2\sigma^2_{x_1C_0}}) \\ &= \frac{1}{0.18\sqrt{2\pi}}exp(\frac{-(x-1.5)^2}{2\cdot 0.18^2}) = 0.188 \end{aligned} \tag{11}$$

$$\begin{aligned} p(x_3|C_0) &= \frac{1}{\sigma_{x_3C_0}\sqrt{2\pi}}exp(\frac{-(x-\mu_{x_3C_0})^2}{2\sigma^2_{x_1C_0}}) \\ &= \frac{1}{0.245\sqrt{2\pi}}exp(\frac{-(x-4.15)^2}{2\cdot 0.245^2}) = 8,85\cdot 10^{-4} \end{aligned} \tag{12}$$

$$\begin{aligned} p(x_4|C_0) &= \frac{1}{\sigma_{x_4C_0}\sqrt{2\pi}}exp(\frac{-(x-\mu_{x_4C_0})^2}{2\sigma^2_{x_1C_0}}) \\ &= \frac{1}{0.12\sqrt{2\pi}}exp(\frac{-(x-5.65)^2}{2\cdot 0.12^2}) = 1.84\cdot 10^{-3} \end{aligned} \tag{13}$$

Using Eq. 7, one can calculate the probability of the object belonging to class 0.

$$\begin{aligned} p(C_0|X) &= p(C_0)p(x_1|C_0)p(x_2|C_0)p(x_3|C_0)p(x_4|C_0) \\ &= 0.4\cdot 0.665\cdot 0.188\cdot 8.85\cdot 10^{-4}\cdot 1.84\cdot 10^{-3} \\ &= 8.12\cdot 10^{-8} \end{aligned} \tag{14}$$

All of the above steps are repeated for class 1 as well.

$$\begin{aligned} p(x_1|C_1) &= \frac{1}{\sigma_{x_1C_1}\sqrt{2\pi}}exp(\frac{-(x-\mu_{x_1C_1})^2}{2\sigma^2_{x_1C_1}}) \\ &= \frac{1}{0.21\sqrt{2\pi}}exp(\frac{-(x-1.7)^2}{2\cdot 0.21^2}) = 1.34\cdot 10^{-3} \end{aligned} \tag{15}$$

$$\begin{aligned} p(x_2|C_1) &= \frac{1}{\sigma_{x_2C_1}\sqrt{2\pi}}exp(\frac{-(x-\mu_{x_2C_1})^2}{2\sigma^2_{x_1C_1}}) \\ &= \frac{1}{0.13\sqrt{2\pi}}exp(\frac{-(x-2.4)^2}{2\cdot 0.13^2}) = 1.88\cdot 10^{-3} \end{aligned} \tag{16}$$

$$\begin{aligned} p(x_3|C_1) &= \frac{1}{\sigma_{x_3C_1}\sqrt{2\pi}}exp(\frac{-(x-\mu_{x_3C_1})^2}{2\sigma^2_{x_1C_1}}) \\ &= \frac{1}{0.103\sqrt{2\pi}}exp(\frac{-(x-2.96)^2}{2\cdot 0.103^2}) = 0.302 \end{aligned} \tag{17}$$

$$p(x_4|C_1) = \frac{1}{\sigma_{x_4C_1}\sqrt{2\pi}}exp(\frac{-(x-\mu_{x_4C_1})^2}{2\sigma^2_{x_1C_1}}) = \frac{1}{0.0214\sqrt{2\pi}}exp(\frac{-(x-0.153)^2}{2\cdot 0.0214^2}) = 0.884 \quad (18)$$

$$p(C_1|X) = p(C_1)p(x_1|C_1)p(x_2|C_1)p(x_3|C_1)p(x_4|C_1) = 0.6\cdot 1.34\cdot 10^{-3}\cdot 1.88\cdot 10^{-3}\cdot 0.302\cdot 0.884 = 8.12\cdot 10^{-8} = 3.845742252\cdot 10^{-7} \quad (19)$$

Given that the posterior probability for class 1 is greater, the Bayes classifier assigns the object X to class 1 based on its feature values.

2.2. Multivariate Gaussian distribution

The multivariate Gaussian distribution [4] is a generalization of the normal distribution to cases where the observed objects have more than one feature. To illustrate how this method works, it is simplest to begin with the one-dimensional case. The probability density function of the one-dimensional normal distribution is given by Eq. 20.

$$\mathcal{N}(x,\mu,\sigma^2) = \frac{1}{\sigma\sqrt{2\pi}}e^{-(x-\mu)^2/2\sigma^2} \quad (20)$$

The probability density function is characterized by two parameters: μ—the expected value, which determines the location of the function's peak, and σ^2—the variance, which describes the sharpness (or spread) of the curve.

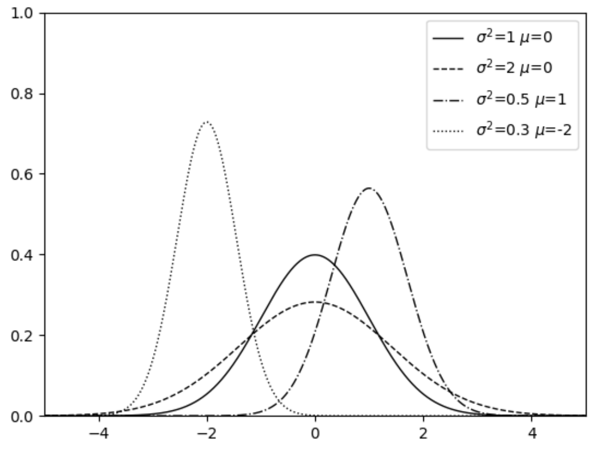

Figure 1: Gaussian distributions with distinct parameters

The principles that apply in the one-dimensional space can be generalized to its n-dimensional counterpart, as expressed by Eq. 21.

$$\mathcal{N}(X,\mu,\Sigma) = \frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}}e^{-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)} \quad (21)$$

As in the one-dimensional case, the probability density function is defined by two parameters, which retain the same interpretation. The key difference lies in the transition from scalar values to matrices. When analyzing an n-dimensional space, the parameters are:

- X—a random vector of size $n\times 1$
- μ—a mean vector of size $n\times 1$
- Σ—a covariance matrix of size $n\times n$

Given a set of k data points, the parameters of the equation are estimated using the maximum likelihood principle [5].

$$\mu = \frac{1}{k}\sum_{i=1}^{k}X_i \quad (22)$$

$$\Sigma = \frac{1}{k}\sum_{i=1}^{k}(X_i-\mu)(X_i-\mu)^T \quad (23)$$

2.2.1 Example

The application of the multivariate Gaussian distribution can be illustrated using an example based on the dataset provided in Tab. 4.

Table 4: Training data

Class	feature x_1	feature x_2
0	2.8	2.6
0	4.0	3.1
0	1.2	2.9
0	0.7	3.3
0	5.1	2.7
1	7.8	9.6
1	5.9	8.1
1	6.6	7.9
1	7.1	6.3
1	6.1	9.7

It should be noted that the above objects have only two features not due to any limitation of the method, but in order to enable visualization of the results.

In Eq 24, Eq. 22 is used to compute the parameter μ for class 0, while in 25, Eq. 23 is applied to calculate Σ for the same class, using the previously computed vectors $(X_i-\mu)$. The next step is to repeat the above procedure for class 1.

$$\mu_0 = \frac{1}{5}(\begin{bmatrix}2.8\\2.6\end{bmatrix}+\begin{bmatrix}4.0\\3.1\end{bmatrix}+\begin{bmatrix}1.2\\2.9\end{bmatrix}+\begin{bmatrix}0.7\\3.3\end{bmatrix}+\begin{bmatrix}5.1\\2.7\end{bmatrix}) = \frac{1}{5}\begin{bmatrix}13.8\\14.6\end{bmatrix} = \begin{bmatrix}2.76\\2.92\end{bmatrix} \quad (24)$$

$$
\begin{aligned}
\Sigma_0 &= \frac{1}{5}\Bigg(\begin{bmatrix}0.06\\-0.32\end{bmatrix}\begin{bmatrix}0.06 & -0.32\end{bmatrix}+\begin{bmatrix}1.24\\0.18\end{bmatrix}\begin{bmatrix}1.24 & 0.18\end{bmatrix}\\
&\quad+\begin{bmatrix}-1.56\\-0.02\end{bmatrix}\begin{bmatrix}-1.56 & -0.02\end{bmatrix}+\begin{bmatrix}-2.06\\0.38\end{bmatrix}\begin{bmatrix}-2.06 & 0.38\end{bmatrix}\\
&\quad+\begin{bmatrix}2.34\\-0.22\end{bmatrix}\begin{bmatrix}2.34 & -0.22\end{bmatrix}\Bigg)\\
&= \frac{1}{5}\begin{bmatrix}13.6936 & -1.0612\\-1.0612 & 0.328\end{bmatrix}=\begin{bmatrix}2.7387 & -0.2122\\-0.2122 & 0.0656\end{bmatrix}
\end{aligned}
\tag{25}
$$

$$
\begin{aligned}
\mu_1 &= \frac{1}{5}\left(\begin{bmatrix}7.8\\9.6\end{bmatrix}+\begin{bmatrix}5.9\\8.1\end{bmatrix}+\begin{bmatrix}6.6\\7.9\end{bmatrix}+\begin{bmatrix}7.1\\6.3\end{bmatrix}+\begin{bmatrix}6.1\\9.7\end{bmatrix}\right)\\
&= \frac{1}{5}\begin{bmatrix}33.5\\41.6\end{bmatrix}=\begin{bmatrix}6.7\\8.32\end{bmatrix}
\end{aligned}
\tag{26}
$$

$$
\begin{aligned}
\Sigma_1 &= \frac{1}{5}\Bigg(\begin{bmatrix}1.2\\1.28\end{bmatrix}\begin{bmatrix}1.2 & 1.28\end{bmatrix}+\begin{bmatrix}-0.8\\-0.22\end{bmatrix}\begin{bmatrix}-0.8 & -0.22\end{bmatrix}\\
&\quad+\begin{bmatrix}-0.1\\-0.42\end{bmatrix}\begin{bmatrix}-0.1 & -0.42\end{bmatrix}+\begin{bmatrix}0.4\\-2.02\end{bmatrix}\begin{bmatrix}0.4 & -2.02\end{bmatrix}\\
&\quad+\begin{bmatrix}0.6\\1.38\end{bmatrix}\begin{bmatrix}0.6 & 1.38\end{bmatrix}\Bigg)\\
&= \begin{bmatrix}0.476 & -0.002\\-0.002 & 1.569\end{bmatrix}
\end{aligned}
\tag{27}
$$

Having calculated the mean and covariance values, we can examine the probability density of belonging to the given class, or the probability itself with which an object belongs to it. To obtain this, it is sufficient to substitute the computed μ, Σ, and the feature vector X of the object to be classified into Eq. 21. This will be calculated in the following subsection. A graphical example of the resulting probability distribution for the dataset shown in Tab. 4 is presented in Fig. 2.

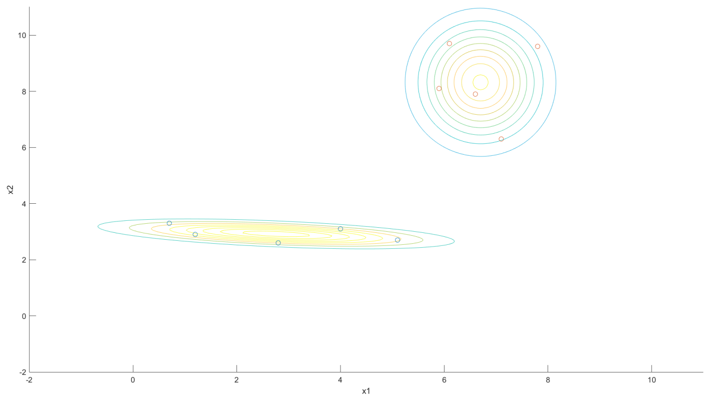

Figure 2: The normal distribution was constructed based on the training dataset. Objects belonging to class 0 are represented by blue points, while those of class 1 are shown in orange.

Table 5: Object to be classified

Class	feature x_1	feature x_2
?	3.2	2.7

$$
\mathcal{N}_0\left(\begin{bmatrix}3.2\\2.7\end{bmatrix},\begin{bmatrix}2.76\\2.92\end{bmatrix},\begin{bmatrix}2.7384 & -0.2112\\-0.2112 & 0.0656\end{bmatrix}\right)=0.294 \tag{28}
$$

$$
\mathcal{N}_1\left(\begin{bmatrix}3.2\\2.7\end{bmatrix},\begin{bmatrix}6.7\\8.32\end{bmatrix},\begin{bmatrix}0.476 & -0.002\\-0.002 & 1.569\end{bmatrix}\right)=1.924\cdot10^{-11} \tag{29}
$$

The above results should be interpreted as the probabilities that the test object belongs to a given class. Eq. 28 shows the probability of belonging to class 0, while Eq. 29 corresponds to class 1. Since these values differ significantly, it can be stated with a high degree of confidence that the object belongs to class −1. As shown in Fig. 3, the graphical interpretation also confirms this conclusion.

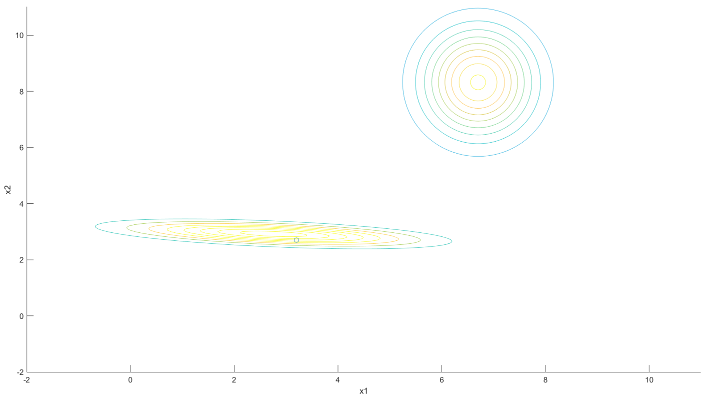

Figure 3: Test object against the probability density

3. Unsupervised Generative Methods

The second type of methods—unsupervised learning—does not use labels. Similar to supervised methods, the algorithm learns relationships between data points from the input data but lacks information about object-class associations. These methods allow for the identification of groups of objects with similar features.

3.1. Gaussian mixture model

The multivariate Gaussian distribution can be useful for modeling multiple classes; however, in reality, data sets often form complex structures and do not always follow a simple normal distribution. One such example is illustrated in Fig. 4.

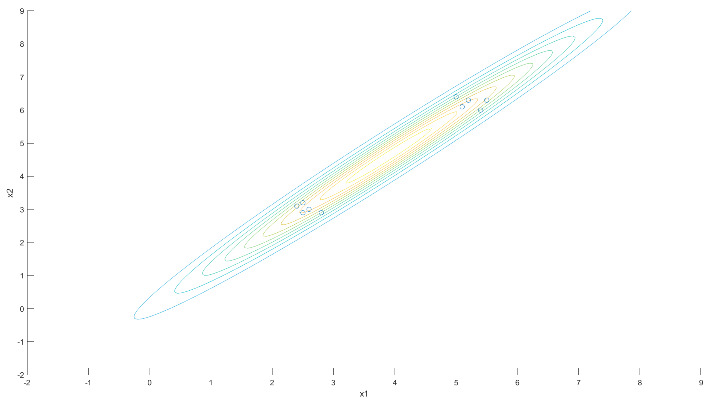

Figure 4: Solution using a single Gaussian distribution

As can be seen, the expected value lies between the two clusters of objects, which is a situation difficult to capture using a model based on a single distribution. A solution to this problem may be to describe the model using two normal distributions [6]. To achieve this effect, the objects need to be divided into two groups, and the distribution parameters calculated separately for each.

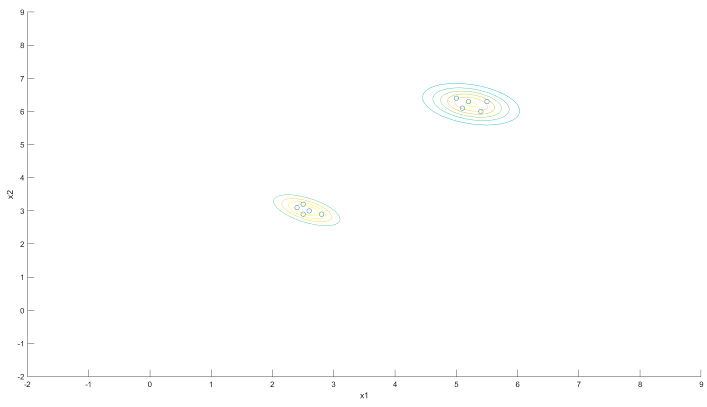

Figure 5: Solution using two Gaussian distributions

Additionally, it is important to remember the fundamental axiom of probability theory, expressed by Eq. 30

$$P(\Omega) = 1 \tag{30}$$

In order for Eq. 30 to be satisfied, it is sufficient to multiply each Gaussian distribution equation by a certain parameter. As a result, when describing a dataset with k equations, an additional k parameters ϕ_k must be introduced which satisfy the condition described by Eq.31.

$$\sum_{i=1}^{k} \phi_i = 1 \tag{31}$$

Summarizing all the aforementioned information, the final formula for the Gaussian Mixture Model method can be written as Eq. (32).

$$\mathscr{G}(X, \phi_n) = \sum_{i=1}^{k} \phi_i \mathscr{N}(X, \mu_i, \Sigma_i) \tag{32}$$

as well as its more complete version described by Eq. 33:

$$\mathscr{G}(X, \phi_n) = \sum_{i=1}^{k} \phi_i \frac{1}{(2\pi)^{n/2}|\Sigma_i|^{1/2}} e^{-\frac{1}{2}(x-\mu_i)^T \Sigma_i^{-1}(x-\mu_i)} \tag{33}$$

3.1.1 Expectation-Maximization Algorithm

Since GMM is an unsupervised method, the input dataset lacks labels, which makes it impossible to calculate the optimal model parameters through empirical risk minimization. To determine the labels that maximize the similarity of objects within a given region, the Expectation-Maximization algorithm [7, 8] can be employed. It consists of two steps: Expectation (E)—calculation of parameters based on the current assignment of objects, and Maximization (M)—assignment of objects to the currently best-fitting cluster.

In the E-step, we estimate the parameters according to the equations described in subsection 2.2 (multivariate Gaussian distribution). The additional parameter ϕ, which is not covered in that subsection, is calculated according to Eq. 34 after the main loop of the algorithm finishes.

$$\phi_i = \frac{\text{number of objects of group i}}{\text{total number of objects}} \tag{34}$$

In the M step, using the newly calculated parameters, each object is evaluated to determine which component distribution it fits best. Simply put, each individual object is evaluated against every component distribution, and if the probability of belonging to a group other than its current one is higher, the object is reassigned to that group by changing its label. Otherwise, its label remains unchanged.

The final consideration is the stopping criterion. The most common approach is to stop the algorithm when no changes occur during the M step in the subsequent iteration. However, if execution time is more important than clustering accuracy, a maximum number of iterations can be set as the stopping condition.

3.1.2 Example

To illustrate the course of the Expectation-Maximization (E-M) algorithm, a dataset forming two distinct groups in a two-dimensional space, shown in Fig. 6, will be used. The initial assignment to groups will be chosen randomly; however, to optimize the process, the initial state is often selected using methods such as the k-means algorithm [9].

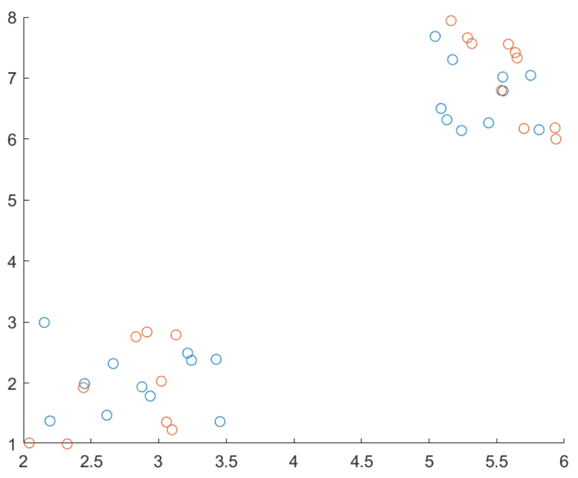

Figure 6: The distribution of points and their group memberships are represented by the colors of the markers.

After completing the initial steps, the main loop of the algorithm can begin. The subsequent charts represent the state after each successive iteration of the algorithm.

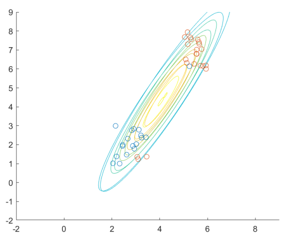

Figure 7: First iteration.

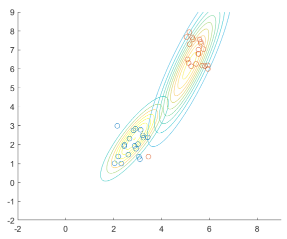

Figure 8: Second iteration.

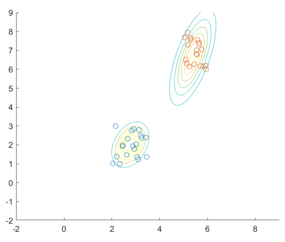

Figure 9: Third iteration.

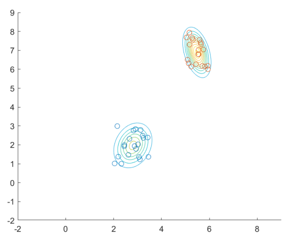

Figure 10: Fourth iteration.

Table 6: Parameter values of the distribution at subsequent iterations

Itertion	μ_1	Σ_1	μ_2	Σ_2
1	$\begin{bmatrix}3.72\\3.82\end{bmatrix}$	$\begin{bmatrix}2.17 & 3.58\\3.58 & 6.84\end{bmatrix}$	$\begin{bmatrix}4.33\\5.11\end{bmatrix}$	$\begin{bmatrix}2.04 & 3.47\\3.47 & 6.23\end{bmatrix}$
2	$\begin{bmatrix}2.98\\2.44\end{bmatrix}$	$\begin{bmatrix}1.03 & 1.23\\1.23 & 2.03\end{bmatrix}$	$\begin{bmatrix}5.12\\6.60\end{bmatrix}$	$\begin{bmatrix}1.04 & 1.70\\1.70 & 3.01\end{bmatrix}$
3	$\begin{bmatrix}2.67\\2.07\end{bmatrix}$	$\begin{bmatrix}0.20 & 0.01\\0.01 & 0.27\end{bmatrix}$	$\begin{bmatrix}5.53\\7.18\end{bmatrix}$	$\begin{bmatrix}0.08 & 0.07\\0.07 & 0.32\end{bmatrix}$
4	$\begin{bmatrix}2.67\\2.07\end{bmatrix}$	$\begin{bmatrix}0.19 & 0.01\\0.01 & 0.28\end{bmatrix}$	$\begin{bmatrix}5.53\\7.18\end{bmatrix}$	$\begin{bmatrix}0.08 & 0.07\\0.07 & 0.32\end{bmatrix}$

As can be seen, within four iterations the algorithm was able to correctly detect and estimate the probability distributions for both groups. However, it should be noted that the selection of points in the space was tailored for visualization purposes. When working with real-world data, the number of iterations required to achieve satisfactory results will be significantly higher, and the distribution itself may not be as clear as shown in the presented charts. The final stage of the algorithm is the estimation of the parameters ϕ. Knowing that there are 20 objects in each group, we obtain:

$\phi_1 = \frac{20}{40} = 0.5$ and $\phi_2 = \frac{20}{40} = 0.5$.

Knowing all the parameters, the final Eq. 35 describing the studied model can be written as follows:

$$\begin{aligned}\mathscr{G}(X,[0.5,0.5]) \quad = \quad & 0.5\mathscr{N}_1(X,\begin{bmatrix}2.80\\1.97\end{bmatrix},\begin{bmatrix}0.17 & 0.05\\0.05 & 0.37\end{bmatrix}) \\ & +0.5\mathscr{N}_2(X,\begin{bmatrix}5.47\\6.89\end{bmatrix},\begin{bmatrix}0.07 & -0.06\\-0.06 & 0.38\end{bmatrix})\end{aligned} \tag{35}$$

A common mistake when learning how the Gaussian Mixture Model (GMM) works is treating it as a simple multivariate normal distribution. It is worth analyzing the Figs. 2 and 5. Both figures show two normal distributions; however, the figure from subsection 2.2 presents one distribution per class, whereas its counterpart in GMM shows two distributions, but not for two classes — rather, both belong to a single class. This is the key difference between the two methods discussed.

3.2. Variational autoencoder

An autoencoder [10] is a neural network model where typically the number of input and output neurons is the same, and the architecture is symmetrical, consisting of an encoding block and a decoding block. Thanks to this structure, the model is capable of reconstructing the input data. Within the network, there is a vector shorter than the input vector, implementing a so-called bottleneck, which results in dimensionality reduction of the input data.

Autoencoders and their variants have a wide range of applications: real-time foreign language translation [11], anomaly detection in data [12], noise removal from images [13], as well as from various waves, signals, and audio files. This subsection will discuss the operating principle of a variant of the autoencoder that enables the generation of new data based on a training set, namely the Variational Autoencoder (VAE) [14].

To explain the functioning of a VAE, we will first clarify how a standard autoencoder operates. As an example, we will use the classification of digit images (0-9), each with a dimension of 28x28 pixels, where each pixel takes a value from 0 (white) to 1 (black). Such an image can be represented as a vector of length 784 elements $(28*28)$, where each element of the vector corresponds to the color value of an individual pixel, as illustrated in Fig. 11.

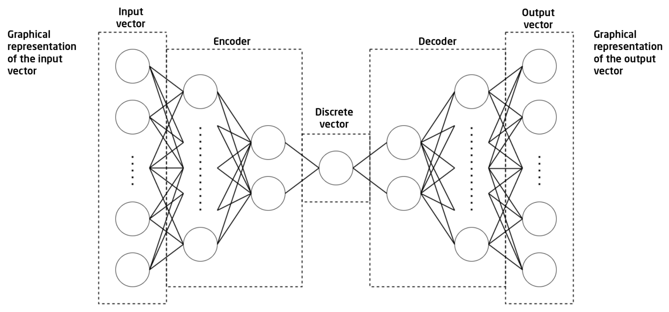
Graphical representation of the input vector
Input vector
Encoder
Discrete vector
Decoder
Output vector
Graphical representation of the output vector

Figure 11: Example architecture of an autoencoder network

In an autoencoder, just like in a classical neural network, each connection between neurons has weights, which are initially set to random values at the start of training. The values of neurons in subsequent layers are equal to the sum of the products of individual neuron values and the weights of the corresponding connections, plus an additional bias term. The example below is illustrated in Fig. 12.

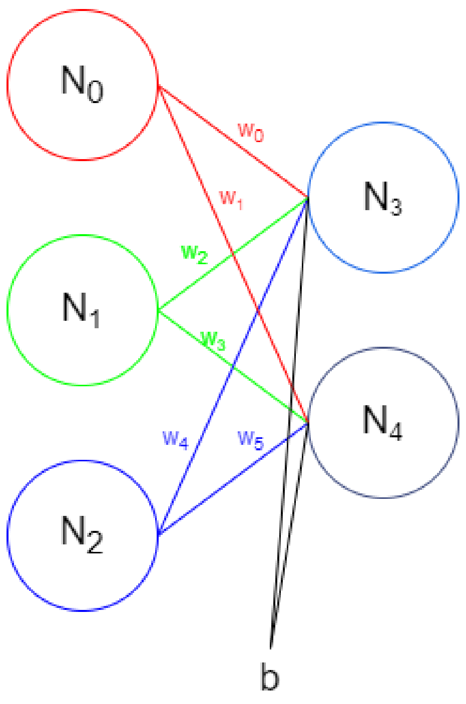

Figure 12: Obtaining a new weight based on the neural network

For data

$$\begin{array}{lll} N_0 = 0.9 & w_0 = 0.1 & w_1 = 0.25 \\ N_1 = 0.5 & w_2 = 0.4 & w_3 = 0.3 \\ N_2 = 0.25 & w_4 = 0.9 & w_5 = 0.5 \\ b = 0.25 & & \end{array}$$

The values of N_3 and N_4 can be calculated using the following equations:

$$N_3 = N_0 * w_0 + N_1 * w_2 + N_2 * w_4 + b \tag{36}$$

$$N_4 = N_0 * w_1 + N_1 * w_3 + N_2 * w_5 + b \tag{37}$$

To obtain the final output values of the neurons in the subsequent layers, an activation function should be applied, for example the sigmoid function: $N_n = \frac{1}{1+e^{-N_n}}$, where n is the index of the respective neuron. For instance, the value of neuron N_3 will be $\frac{1}{1+e^{0.9*0.1+0.5*0.4+0.25}} \approx 0.37$. Fig. 11 shows the architecture of the Variational Autoencoder (VAE). The entire described process corresponds to the encoder and decoder operations, with a reduced vector located in the middle of the system. After training the network, any vector of the appropriate size can be input in this middle section; however, this only allows for the classification of an object, since only a single object is being tested.

To generate a new object, instead of a fixed vector in the middle of the system, a distribution of its parameters is chosen. As a result, the network architecture looks somewhat different, as shown in Fig. 13.

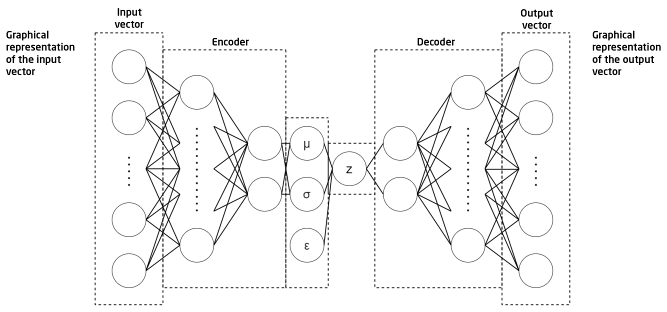
Graphical representation of the input vector
Input vector
Encoder
μ
σ
ε
z
Decoder
Output vector
Graphical representation of the output vector

Figure 13: Diagram of the VAE architecture

Unlike a classical autoencoder, the middle part of the Variational Autoencoder (VAE) architecture applies the so-called Reparametrization Trick [15]. This trick allows the decoded vector to be transformed from a deterministic form into a stochastic one, while retaining some determinism. This enables the generation and recognition of objects spanning a discrete spectrum of parameter values.

The parameters μ and σ represent the mean and standard deviation vectors of the normal distribution in the encoder layer. Using the sum of the mean values and a randomly sampled standard deviation for each element, the explicit latent vector z is created as described by Eq. 38.

$$z = \mu + \sigma \odot \varepsilon \tag{38}$$

where:
μ—the vector of mean values of the parameters of the re-

duced vector,
$\sigma \odot \varepsilon$—the vector of standard deviations of the parameters multiplied element-wise by a random value drawn from the standard normal distribution $\mathcal{N}(0,1)$. Thanks to training the network using the distribution of parameter values, the system can learn to generate new, similar but entirely synthetic objects.

In the previously mentioned example of generating new handwritten digit images, for better visualization the vector z contains two parameters. During the creation of this vector, it can take any values; however, only those marked on the plot will have meaningful results.

Fig. 14 shows the distribution of the generated digit images. In Figs. 15 and 16, example objects are shown. The first one—15—belongs to the cluster of ones, and therefore its shape clearly resembles the digit 1. The second object—16—does not lie near any cluster, which results in a shape that does not resemble any digit.

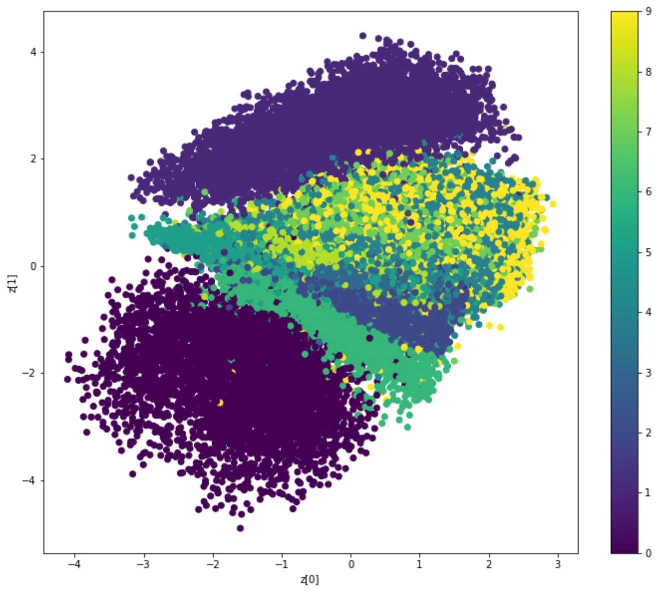

Figure 14: Distribution of class objects

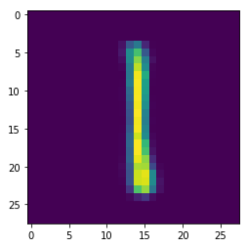
Figure 15: $z=[1,4]^T$ is located within the cluster of ones and clearly represents the digit 1

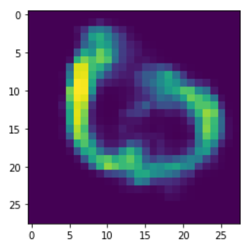
Figure 16: $z=[2,-4]^T$ lies in an area without any cluster; the closest resemblance it has is to the digit 0

A classical autoencoder is trained using the backpropagation algorithm, which typically enables the detection of a given class of objects, as previously mentioned, due to the selection of a single vector. Thanks to the reparametrization trick, training of the VAE can be conducted to generate new objects, with the difference that the error between the output vector and the target is calculated using the Kullback-Leibler divergence [16], which measures how much a given distribution deviates from the standard normal distribution of the compared object.

$$\sum_{i=1}^{n} p_\theta(i) \log_2 \frac{p_\theta(i)}{p_\omega(i)} \tag{39}$$

where:
n—the number of parameters of the trained vector
$p_\theta(i)$—the probability from the training model
$p_\omega(i)$—the probability from the learned model
Let's take an example of two vectors—the output vector $p_\omega = \begin{bmatrix}0.35\\0.56\end{bmatrix}$ and the training vector $p_\theta = \begin{bmatrix}0.28\\0.72\end{bmatrix}$. Using Eq. 39, we can calculate the Kullback-Leibler divergence by substituting the respective values from both vectors.

$$\sum_{i=1}^{n} p_\theta(i) \log_2 \frac{p_\theta(i)}{p_\omega(i)} = 0.28 * \log_2 \frac{0.28}{0.35} + 0.72 * \log_2 \frac{0.72}{0.56} \approx 0.17 \tag{40}$$

In the case of VAE, the vector z (more precisely, the parameters of its normal distribution) is compared to the standard normal distribution $\mathcal{N}(0,1)$.

$$\sum_{i=1}^{n} \sigma_i^2 + \mu_i^2 - log(\sigma_i) - 1 \tag{41}$$

where:
n—the number of parameters in the learned vector
σ_i—the i-th standard deviation of the parameter
μ_i—the i-th mean value of the parameter
The influence of variables on the shape of the studied and normal distributions is illustrated in Figs. 17 and 18. These figures present distribution plots for various values of the mean, standard deviation, and KL Divergence.

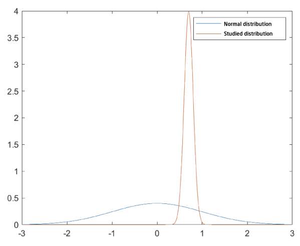

Figure 17: σ=0.7, μ=0.1, KL Divergence = 0.5

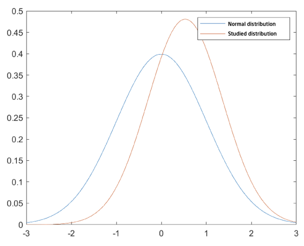

Figure 18: σ=0.53, μ=0.83, KL Divergence = 0.05

As shown in Figs. 17 and 18, the closer the distribution of the encoded vector is to the normal distribution, the smaller the divergence. This divergence constitutes a component of the neural network's training objective function.

4. Conclusion

This paper presents the application of generative models to categorization tasks. At the outset, a distinction was made between two learning paradigms: supervised and unsupervised learning. Subsequently, a basic supervised generative algorithm—the Naive Bayes classifier—was described. This example illustrates the fundamental principles of supervised methods, namely learning class distributions, and serves to demonstrate how generative classifiers assign new objects to classes. Thanks to its fast execution and simple implementation, especially in cases where class features are independent, the Bayes algorithm is frequently used in practice in text-related classifiers, such as spam filters.

The next subsection introduces a more complex model—the multivariate Gaussian distribution. It leverages a property discovered by the German scientist Carl Friedrich Gauss, which states that any variable that is the sum of many independent factors tends to follow a distribution resembling the bell curve, commonly known as the normal distribution. The numerical example presented earlier in this paper demonstrated that given a test dataset, it is possible to construct a probabilistic model based on the normal distribution, enabling us to associate newly added points with classes with high probability.

The second part of the paper describes unsupervised models. The first section covers the Gaussian Mixture Model (GMM)—an extension of the previously discussed multivariate Gaussian distribution. Their practical applications are very similar; however, GMM allows for a much better approximation of real-world distributions by decomposing them into multiple normal distributions. A drawback of this method is the inability to analytically determine the parameters of the model. Therefore, the Expectation-Maximization (EM) algorithm is also described, which enables iterative approximation of these parameter values. It should be noted that the EM algorithm is computationally expensive, and if the number of iterations is limited, the results obtained may be unsatisfactory. Hence, it is crucial to analyze the input data and decide which method to use: the faster but less accurate multivariate Gaussian distribution, or the slower but more precise GMM.

The final part of the paper is devoted to the Variational Autoencoder—a variant of autoencoder neural networks. A significant advantage of this approach is its wide range of applications, including noise reduction, generation of new objects, interpolation, and watermark removal. Its implementation is relatively straightforward, partly thanks to Python libraries such as Keras and TensorFlow. A drawback of the Variational Autoencoder, compared to other methods (e.g., GAN [17]), is that it generates less precise results. VAE is useful for generating data to train machine learning algorithms rather than serving as a generator itself. For example, images generated by VAE tend to be blurry compared to those produced by GAN.

References

[1] T. Jebara, *Machine learning: discriminative and generative*, vol. 755. Springer Science & Business Media, 2012.

[2] D. Berrar, "Bayes' theorem and naive bayes classifier," *Encyclopedia of Bioinformatics and Computational Biology: ABC of Bioinformatics*, vol. 403, 2018.

[3] K. M. Leung, "Naive bayesian classifier," *Polytechnic University Department of Computer Science/Finance and Risk Engineering*, vol. 2007, pp. 123–156, 2007.

[4] Y. L. Tong, *The multivariate normal distribution*. Springer Science & Business Media, 2012.

[5] T. W. Anderson, "Maximum likelihood estimates for a multivariate normal distribution when some observations are missing," *Journal of the american Statistical Association*, vol. 52, no. 278, pp. 200–203, 1957.

[6] D. A. Reynolds, "Gaussian mixture models.," *Encyclopedia of biometrics*, vol. 741, no. 659-663, 2009.

[7] T. K. Moon, "The expectation-maximization algorithm," *IEEE Signal processing magazine*, vol. 13, no. 6, pp. 47–60, 1996.

[8] S. K. Ng, T. Krishnan, and G. J. McLachlan, "The em algorithm," in *Handbook of computational statistics*, pp. 139–172, Springer, 2012.

[9] K. P. Sinaga and M.-S. Yang, "Unsupervised k-means clustering algorithm," *IEEE access*, vol. 8, pp. 80716–80727, 2020.

[10] D. Bank, N. Koenigstein, and R. Giryes, "Autoencoders," *arXiv preprint arXiv:2003.05991*, 2020.

[11] S. Chandar AP, S. Lauly, H. Larochelle, M. Khapra, B. Ravindran, V. C. Raykar, and A. Saha, "An autoencoder approach to learning bilingual word representations," *Advances in neural information processing systems*, vol. 27, 2014.

[12] Z. Chen, C. K. Yeo, B. S. Lee, and C. T. Lau, "Autoencoder-based network anomaly detection," in *2018 Wireless telecommunications symposium (WTS)*, pp. 1–5, IEEE, 2018.

[13] L. Yasenko, Y. Klyatchenko, and O. Tarasenko-Klyatchenko, "Image noise reduction by denoising autoencoder," in *2020 IEEE 11th International Conference on Dependable Systems, Services and Technologies (DESSERT)*, pp. 351–355, IEEE, 2020.

[14] D. P. Kingma, M. Welling, *et al.*, "An introduction to variational autoencoders," *Foundations and Trends® in Machine Learning*, vol. 12, no. 4, pp. 307–392, 2019.

[15] O. Fabius and J. R. Van Amersfoort, "Variational recurrent autoencoders," *arXiv preprint arXiv:1412.6581*, 2014.

[16] J. M. Joyce, "Kullback-leibler divergence," in *International encyclopedia of statistical science*, pp. 720–722, Springer, 2011.

[17] A. Creswell, T. White, V. Dumoulin, K. Arulkumaran, B. Sengupta, and A. A. Bharath, "Generative adversarial networks: An overview," *IEEE signal processing magazine*, vol. 35, no. 1, pp. 53–65, 2018.